MATH337: Changepoint Detection

Gaetano Romano

2024-11-01

Table of contents

Preface

These are the notes for **MATH337 Changepoint Detection**. They were written by Gaetano Romano.

The module will introduce you to changepoint detection, detailing some algorithms, developing the basics theoretical foundations, and practicing few real-world scenarios.

Across five weeks we will cover the following topics:

1. An introduction to changepoint detection and the CUSUM statistics
2. Controlling the CUSUM and some additional models
3. Dealing with multiple changes
4. PELT, WBS and Penalty selection
5. Working with Real World data.

We will be using R as the programming language for this module. If you're unfamiliar with it, make sure you cover the first three weeks of MATH245.

Every week, you are expected to follow two lectures, one workshop, and one computer aided lab. Over the lecture, we will cover the basics concepts of changepoint detection.

At the end of each chapter, you will find exercises that will be carried in the workshop and the lab. During the workshop, you will be dealing with computations and details about the methodologies, and, finally, during the lab sessions, you'll give a go at programming the various algorithms and running real-world examples.

You will find the solutions to the exercises on the Moodle page, released weekly. If you cannot access the Moodle page, and you still would like to have these solutions, please get in touch with me.

Source files, and attributions

The materials in this course are based on and share elements with the following resources:

- Fearnhead, P., & Fryzlewicz, P. (2022). Detecting a single change-point. *arXiv preprint arXiv:2210.07066*.
- Rebecca Killick's Introduction to Changepoint Detection - a half-day introductory course on changepoint detection.
- Rebecca Killick's Further Changepoint Topics - an extended course on changepoint detection.
- Toby Hocking's Course on Unsupervised Learning, which includes changepoint detection.

I would like to express my gratitude to the authors of these resources. In addition, materials were sourced from various academic papers, which are referenced throughout the body of these notes.

1 An Introduction to Changepoint Detection

1.1 Introduction to Time Series

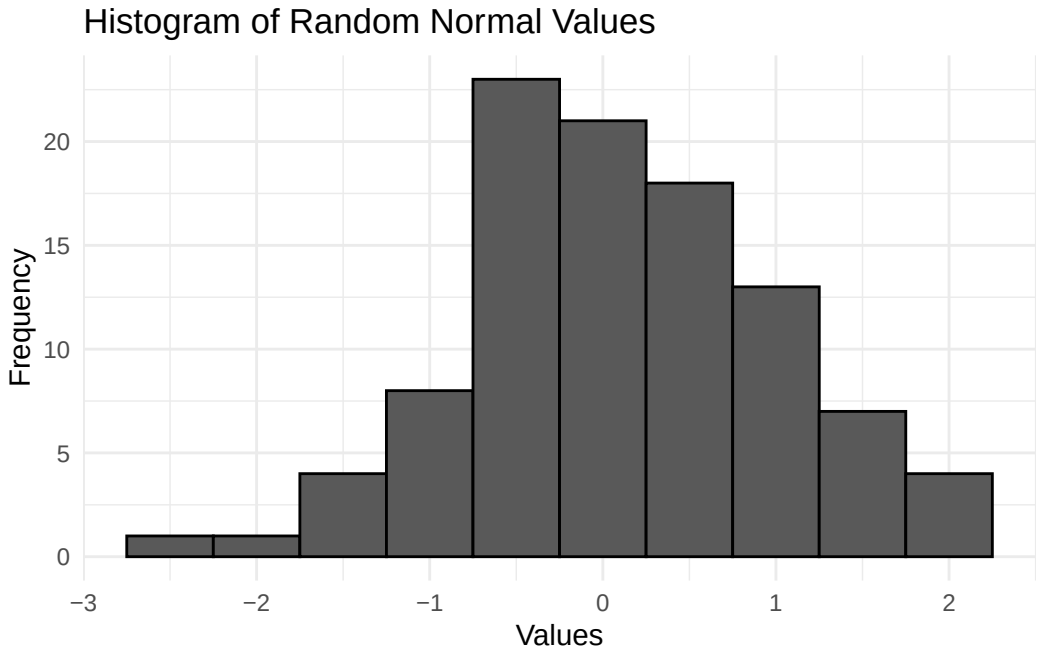
In this module, we will be dealing with **time series**. A time series is a sequence of observations recorded over time (or space), where the order of the data points is crucial.

1.1.1 What is a time series?

In previous modules, such as Likelihood Inference, we typically dealt with data that was not ordered in a particular way. For example, we might have worked with a sample of independent Gaussian observations, where each observation is drawn randomly from the same distribution. This sample might look like the following:

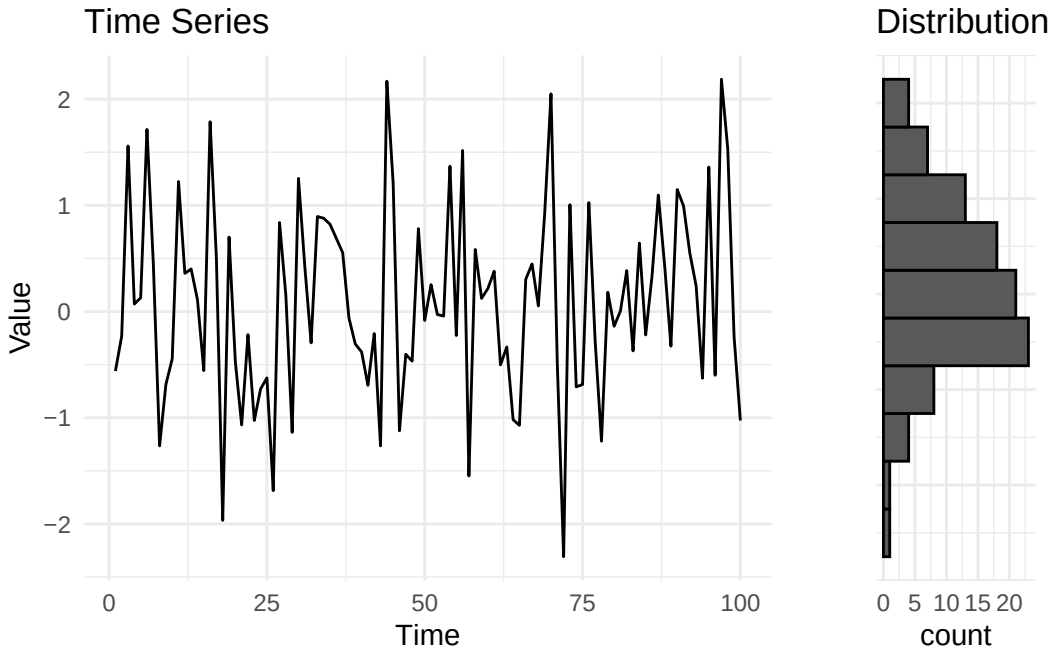
$$y_i \sim \mathcal{N}(0, 1),\ i = 1, \ldots, 100$$

Here, y_i represents the i-th observation, and the assumption is that all observations are independent and identically distributed (i.i.d.) with a mean of 0 and variance of 1.



In this case, the observations do not have any particular order, and our primary interest may be in estimating parameters such as the mean, variance, or mode of the distribution. This is typical for traditional inference, where the order of observations is not of concern.

However, a **time series** involves a specific order to the data—usually indexed by time, although it could also be by space or another sequential dimension. For example, we could assume that the Gaussian sample above is a sequential process, ordered by the time we drew an observation. Each observation corresponds to a specific time point t.



Formal Notation. In time series analysis, use an index t to represent time or order on a given set of observations. The time series vector is written as:

$$y_{1:n} = (y_1, y_2, \dots, y_n).$$

Here, n is the total length of the sequence, and y_t represents the observed value at time t, for $t = 1, 2, \dots, n$. In our previous example, for instance, $n = 100$.

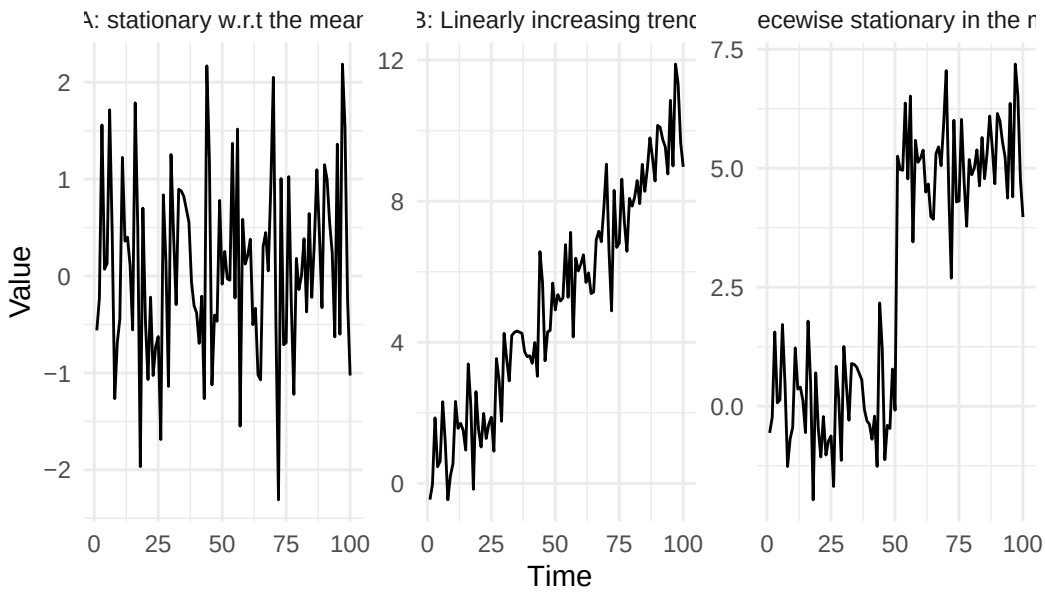
Often, we are also interested in subsets of a time series, especially when investigating specific "windows" or "chunks" of the data. A subset of a time series, starting from time l to time u, with $s \leq u$, will be denoted by the following:

$$y_{l:u} = (y_l, y_{l+1}, \dots, y_u),$$

Where if $l = u$, $y_{l:l} = (y_l)$.

1.1.2 Properties of time series

Time series can have various statistical properties that explain how they behave over time, and they can be characterized based on those. Let us look at three examples of time series:



A. The leftmost time series, was generated by sampling random normal variables $y_t = \epsilon_t,\ \epsilon_t \sim \mathcal{N}(0,1)$. In this case:

$$\mathbb{E}(y_t) = \mathbb{E}(\epsilon_t) = 0,\ \mathrm{Var}(y_t) = \mathrm{Var}(\epsilon_t) = 1,\ \forall\ t \in \{1, ..., 100\}.$$

Say we generate more observations under the same random process, this will give us still a value that will be centered on 0, with variance 1, e.g. $\mathbb{E}(y_{150}) = 0,\ \mathrm{Var}(y_{150}) = 1$.

B. In the centre time series, the series is generated as:

$$y_t = \epsilon_t + 0.1 \cdot t,\ \epsilon_t \sim \mathcal{N}(0,1).$$

This creates a time series with a linear upward trend. Similarly to what done before:

$$\mathbb{E}(y_t) = \mathbb{E}(\epsilon_t) + \mathbb{E}(0.1 \cdot t) = 0.1 \cdot t.$$

Again, saying that we wish to predict the behaviour of the time series at time 150, we know this will be centered on $\mathbb{E}(y_{150}) = 1.5$ (and with which variance?).

C. In the rightmost example, the time series was generated for the first half of the observations as in A., however after $t = 50$, a sudden shift occurs. Mathematically:

$$y_t = \begin{cases} \epsilon_t & \text{for } t \leq 50 \\ \epsilon_t + 5 & \text{for } t > 50 \end{cases}, \quad \epsilon_t \sim \mathcal{N}(0,1)$$

This abrupt change at $t = 50$ introduces a piecewise structure to the data, where the data is seen following a distribution prior to the change, $y_t \sim N(0, 1)$ up to a certain time point $t = 50$, and $y_t \sim N(5, 1)$ after. In many examples of this module, we will be studying processes that are piecewise stationary in the mean and variance, as in this example.

Stationarity in the mean and variance. A time series is said to be *stationary* in mean and variance, if its mean and variance are constant over time. That is, for a time series $y_{1:n}$:

$$\mathbb{E}(y_t) = \mu \quad \text{and} \quad \text{Var}(y_t) = \sigma^2 \quad \forall \in \{1, ..., n\}$$

Similarly, a time series is *non-stationary* in the mean and variance if those change over time.

Piecewise stationary in the mean and variance. A *piecewise stationary* time series is a special case of a non-stationary time series. We will say that a time series is *piecewise stationary* in mean and variance if it is stationary within certain segments but has changes in the mean or variance at certain points, known as *changepoints*. After each changepoint, the series may have a different mean, variance, or both.

Back to our example.

- In A., we can see, very simply how, in this case

  $$\mathbb{E}(y_t) = \mathbb{E}(\epsilon_t) = 0, \forall t \in \{1, ..., 100\},$$

  therefore our series is stationary in the mean and variance.

- In B, we notice that:

  $$\forall t_1, t_2 \in \{1, ..., 100\}, t_1 \neq t_2 \rightarrow \mathbb{E}(y_{t_1}) \neq \mathbb{E}(y_{t_2}).$$

  We can therefore say that the series is non-stationary in the mean.

- In C, $E[y_t] = E[\epsilon_t] = 0$ for $t \leq 50$, and $E[y_t] = E[\epsilon_t] + E[3] = 5$ for $t > 50$. The series is therefore piecewise stationary in the mean.

1.2 Introduction to changepoints

Changepoints are sudden, and often unexpected, shifts in the behavior of a process. They are also known as breakpoints, structural breaks, or regime switches. The detection of changepoints is crucial in understanding and responding to changes in various types of time series data.

The primary objectives in detecting changepoints include:

- **Has a change occurred?**: Identifying if there is a shift in the data.
- **If yes, where is the change?**: Locating the precise point where the change happened.
- **What is the difference between the pre and post-change data?** This may reveal the type of change, and it could indicate differences in parameter values before and after the change.
- **How certain are we of the changepoint location?**: Assessing the confidence in the detected changepoint.
- **How many changes have occurred?**: Identifying multiple changepoints and analyzing each one for similar characteristics.

Changepoints can be found in a wide range of time series, not limited to physical, biological, industrial, or financial processes, and which objectives to follow depends on the type of the analysis we are carrying.

In changepoint detection, there are two main approaches: **online** and **offline** analysis. In applications that require **online analysis**, the data is processed as it arrives, or in small batches. The primary goal of online changepoint detection is to identify changes as quickly as possible, making it crucial in contexts such as process control or intrusion detection, where immediate action is necessary.

On the other hand, **offline analysis** processes all the data at once, typically after it has been fully collected. The aim here is to provide an accurate detection of changepoints, rather than a rapid one. This approach is common in fields like genome analysis or audiology, where the focus is on understanding the structure of the data post-collection.

To give few examples:

1. **Spectroscopy data.** Changepoint detection is useful in spectroscopy data to segment time series of electron emissions into regions of approximately constant intensity, accounting for large-scale fluctuations in laser power and beam pointing.

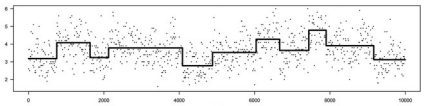


Figure 1.1: Electron emission spectroscopy data, Frick, K., Munk, A., & Sieling, H. (2014).

2. **ECG**: Detecting changes or abnormalities in electrocardiogram (ECG) data can help in diagnosing heart conditions.

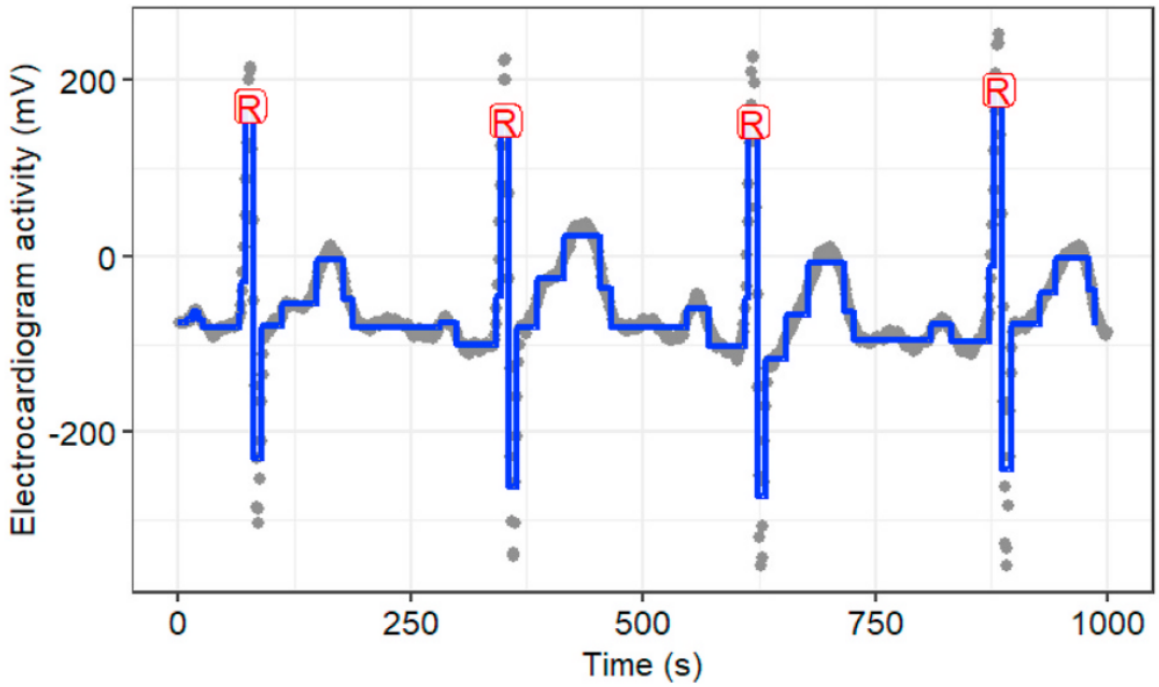


Figure 1.2: Electrocardiograms (heart monitoring), Fotoohinasab et al, Asilomar conference 2020.

3. **Cancer Diagnosis**: Identifying breakpoints in DNA copy number data is important for diagnosing some types of cancer, such as neuroblastoma. This is a typical example of an offline analysis.

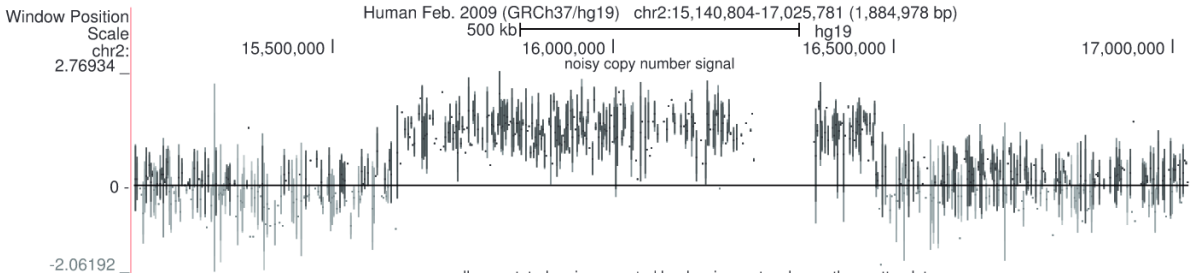


Figure 1.3: DNA copy number data, breakpoints associated with aggressive cancer, Hocking et al, Bioinformatics 2014.

4. **Engineering Monitoring**: Detecting changes in CPU monitoring data in servers can help in identifying potential issues or failures: this is often analysed in real-time on with online methods, with the aim of detecting an issue as quickly as possible.

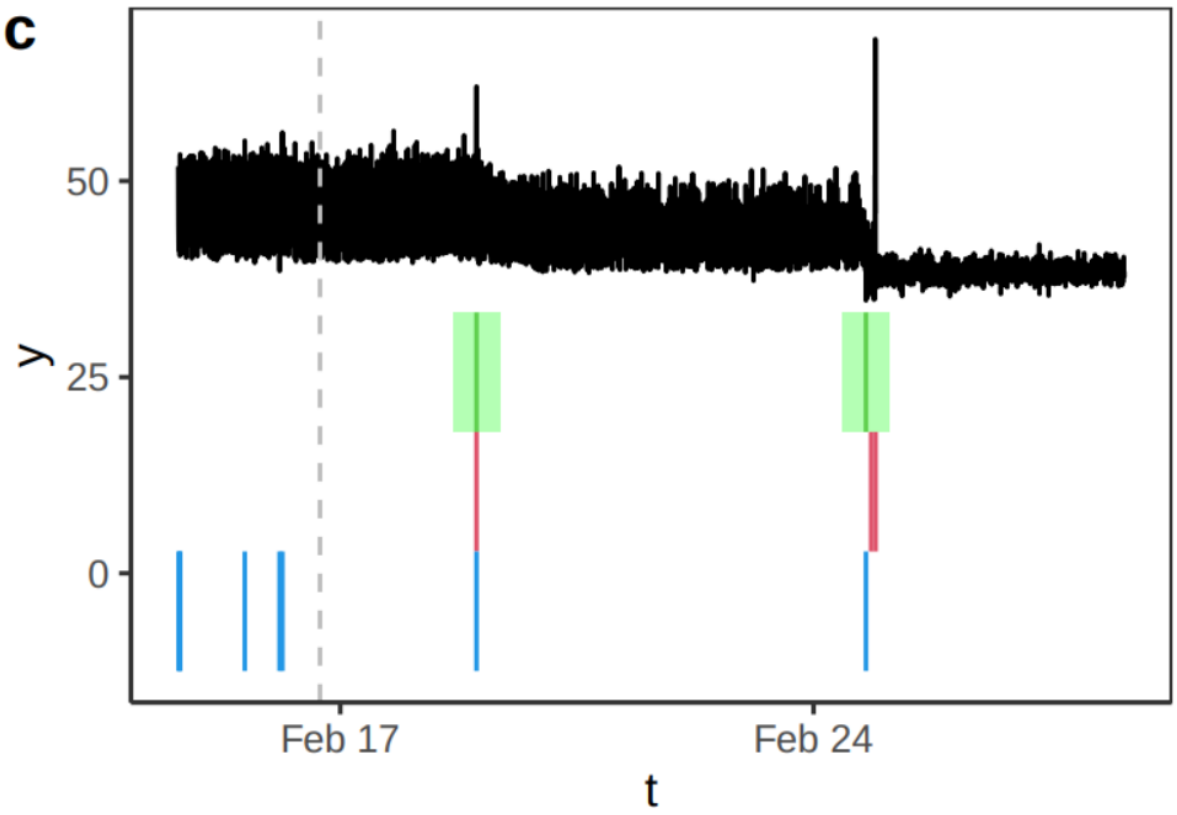
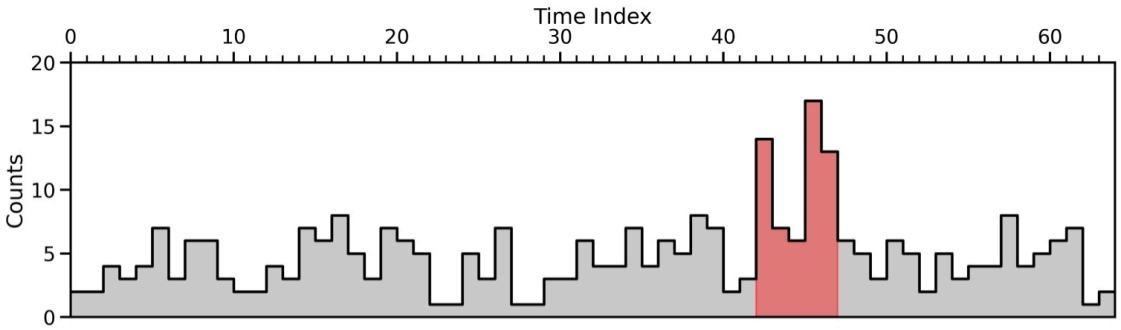


Figure 1.4: Temperature data from a CPU of an AWS server. Source Romano et al., (2023)

5. **Gamma Ray-Burst detection.** Efficient online changepoint detection algorithms can detect gamma-ray bursts from gamma-ray counts on satellites in space. These bursts events happen in just a fraction of a second, and are related to supernova implosions.

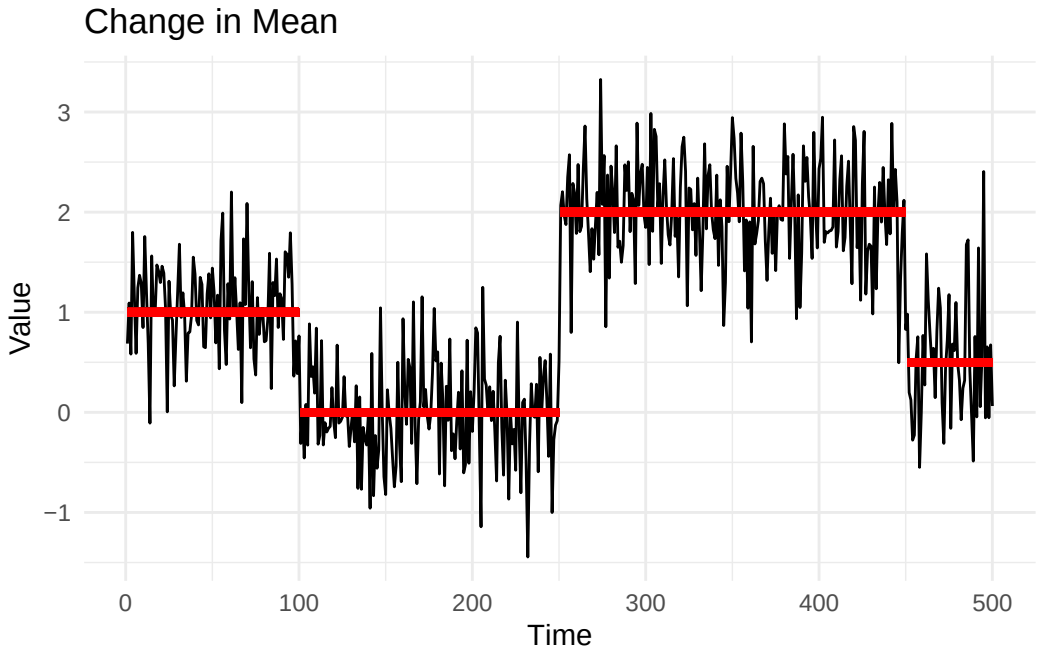


In this module, we will focus exclusively on **offline** changepoint detection, where we assume that all the data is available for analysis from the start.

1.2.1 Types of Changes in Time Series

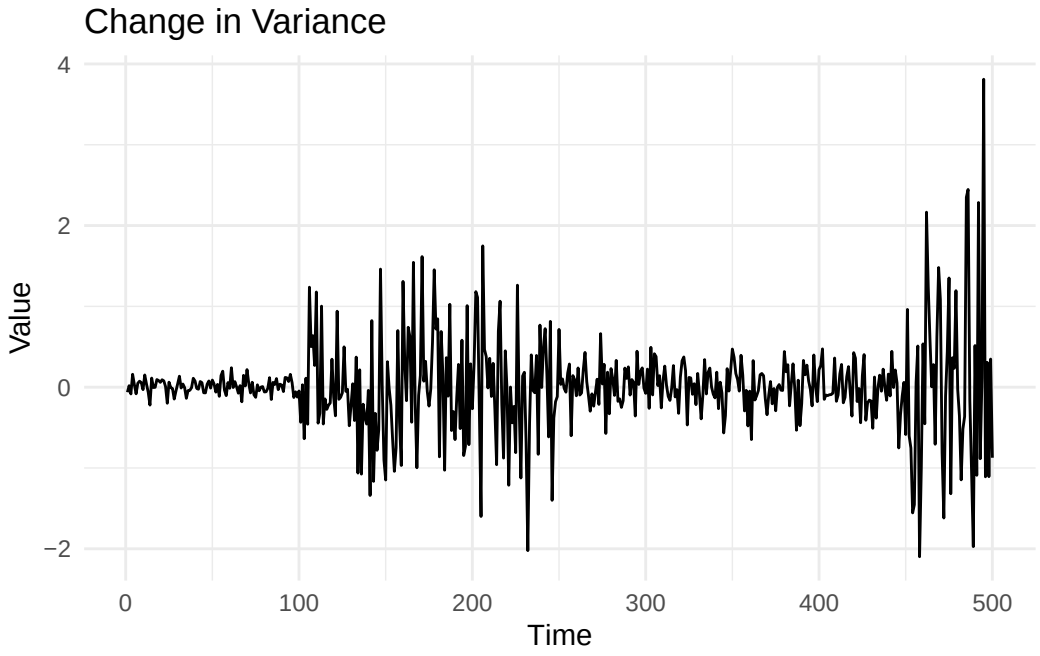
Depending on the model, we could seek for different types of changes in the structure of a time series. Some of the most common types of changes include shifts in mean, variance, and trends in regression. For example, the CPU example above exihibited, in addition to some extreme observations, both changes in mean and variance.

- A **change in mean** occurs when the average level of an otherwise stationary time series shifts from one point to another.



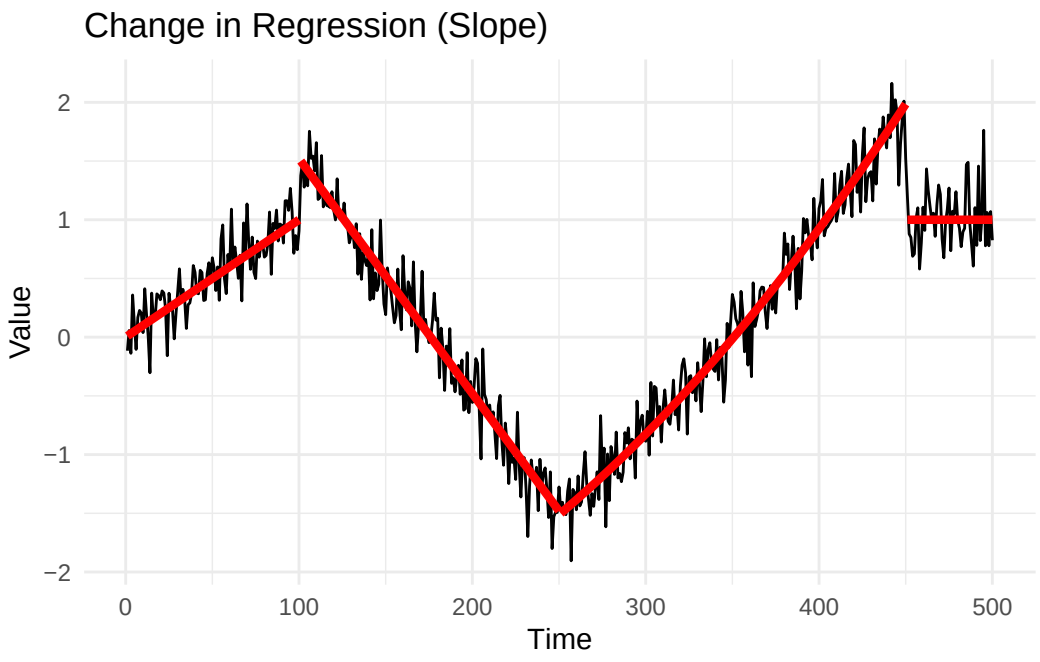
In the plot above, the red lines indicate the true mean values of the different segments.

- A **change in variance** refers to a shift in the variability of the time series data, even when the mean remains constant.



1.2.1.1 3. Change in Regression (Slope)

A **change in regression** or slope occurs when the underlying relationship between time, and/or other auxiliary variables, and the values of the time series changes.

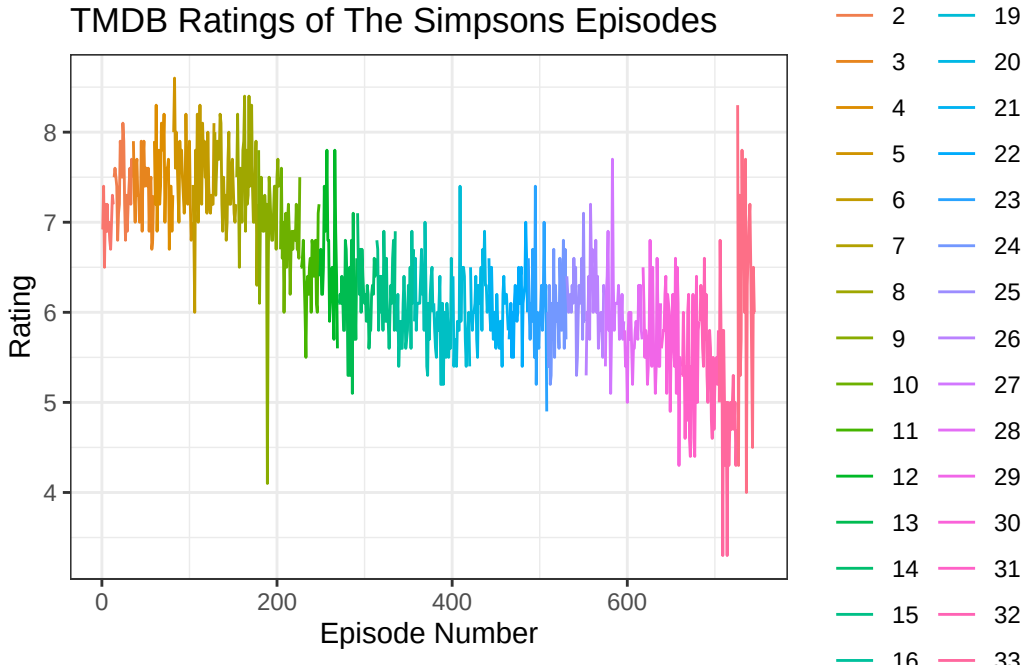


1.2.2 The biggest data challenge in changepoint detection

One of the most widely debated and difficult data challenges in changepoint detection may not be in the field of finance, genetics, or climate science—but rather in television history. Specifically, the question that has plagued critics and fans alike for years is: **At which episode did "The Simpsons" start to decline?**

It's almost common knowledge that "The Simpsons," the longest-running and most beloved animated sitcom, experienced a significant drop in quality over time. But pinpointing exactly *when* this drop occurred is the real challenge. Fortunately, there's a branch of statistics that was practically built to answer questions like these!

I have downloaded a dataset (Bown 2023) containing ratings for every episode of "The Simpsons" up to season 34. We will analyze this data to determine if and when a significant shift occurred in the ratings, which might reflect the decline in quality that so many have observed.



In this plot, each episode of "The Simpsons" is represented by its TMBD rating, and episodes are colored by season. By visually inspecting the graph, we may already start to see some potential points where the ratings decline. However, the goal of our changepoint analysis is to move beyond visual inspection and rigorously detect the exact moment where a significant shift in the data occurs.

Jokes apart, this is a challenging time series! First of all, there's not a clear single change, but rather an increase, followed by a decline. After which, the sequence seems rather stationary. For this reason, throughout the module, we will use this data as a running example to develop

our understanding of various methods, hopefully trying to obtain a definitive answer towards the final chapters. But let's proceed with order…

1.3 Detecting one change in mean

In this section, we will start by exploring the simplest case of a changepoint detection problem: **detecting a change in the mean** of a time series. We assume that the data is generated according to the following model:

$$y_t = \mu_t + \epsilon_t, \quad t = 1, \dots, n,$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ represents Gaussian noise with mean 0 and known variance σ^2, and $\mu_t \in \mathbb{R}$ is the signal at time t, with $\mathbb{E}(y_t) = \mu_t$. The vector of noise terms $\epsilon_{1:n}$ is often referred to as Gaussian noise, and hence, this model is known as *the signal plus noise model*, where the signal is given by $\mu_{1:n}$ and the noise by $\epsilon_{1:n}$.

In *the single change-in-mean problem*, our goal is to determine whether the signal remains constant throughout the entire sequence, or if there exists a point τ, where the mean shifts. In other words, we are testing whether

$$\mu_1 = \mu_2 = \cdots = \mu_n \quad \text{(no changepoint)},$$

or if there exists a time τ such that

$$\mu_1 = \mu_2 = \cdots = \mu_\tau \neq \mu_{\tau+1} = \cdots = \mu_n \quad \text{(changepoint at } \tau\text{)}.$$

Note. The point τ is our *changepoint*, e.g. the first point after which our mean changes, however there's a lot of inconsistencies on the literature: sometimes you will find that people refer to $\tau+1$ as the changepoint, and τ as the last pre-change point (as a matter of fact, please let me know if you spot this inconsistency anywhere in these notes!).

To address this problem, one of the most widely used methods is *the CUSUM (Cumulative Sum) statistic*. The basic idea behind the CUSUM statistic is to systematically compare the mean of the data to the left and right of each possible changepoint τ. By doing so, we can assess whether there is evidence of a significant change in the mean at a given point.

1.3.1 The CUSUM statistics

The CUSUM statistic compares, for a fixed $\tau \in \{1, \dots, n-1\}$, the empirical mean (average) of the data to the left (before τ) with the empirical mean of the data to the right (after τ):

$$C_\tau = \sqrt{\frac{\tau(n-\tau)}{n}} \left| \bar{y}_{1:\tau} - \bar{y}_{(\tau+1):n} \right|,$$

Our $\bar{y}_{1:\tau}$ and $\bar{y}_{(\tau+1):n}$ are just the empirical means of each segment, simply computed with:

$$\bar{y}_{l:u} = \frac{1}{u-l+1} \sum_{t=l}^{u} y_t.$$

The term on the left of the difference, is there to re-scale it so that our statistics is the absolute value of normal random variable that has variance σ^2. If there is no change at τ, this difference is going to be distributed as a standard normal.

This approach is intuitive because if the mean μ is the same across the entire sequence, the values of the averages on both sides of any point τ should be similar. However, if there is a large-enough change in the mean, the means will differ significantly, highlighting the changepoint.

More formally, we declare a change at τ if:

$$\frac{C_\tau^2}{\sigma^2} > c,$$

where the $c \in \mathbb{R}^+$ is a suitable chosen threshold value (in fact it is often chosen as in hypothesis testing).

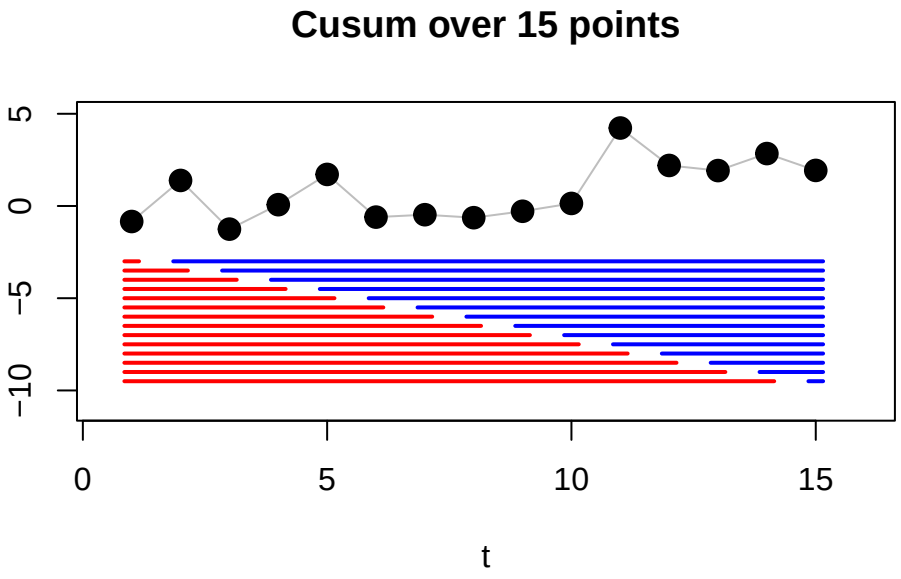
1.3.2 Searching for all τs

In practice, however, we do not know the changepoint location in advance. Our goal is to detect whether a changepoint exists and, if so, estimate its location. To achieve this, we need to consider all possible changepoint locations and choose the one that maximizes our test statistic.

The natural extension of the CUSUM to this situation is to use as a test statistic the maximum of C_τ as we vary τ:

$$C_{max}^2 = \max_{\tau \in \{1,\dots,n-1\}} C_\tau^2/\sigma^2.$$

And detect a changepoint if $C_{max}^2 > c$ for some suitably chosen threshold c. The choice of c will determine the significance level of the test (we'll discuss this in more detail later). Graphically, the test will look as follows:



If we detect a changepoint (i.e., if $C_{max}^2 > c$), we can estimate its location by:

$$\hat{\tau} = \arg \max_{\tau \in \{1,\ldots,n-1\}} C_\tau^2.$$

In other words, $\hat{\tau}$ is the value of τ that maximizes the CUSUM statistic.

A simple estimate of the size of the change is then given by:

$$\Delta\hat{\mu} = \bar{y}_{(\hat{\tau}+1):n} - \bar{y}_{1:\hat{\tau}}.$$

This estimate represents the difference between the mean of the data after the estimated changepoint and the mean of the data before the estimated changepoint.

1.3.3 Example

Let us compute the cusum for the vector $y_{1:4} = (0.5, -0.1, 12.1, 12.4)$.

We know that $n = 4$ (the total number of observations), therefore possible changepoints are: $\tau = 1, 2, 3$.

Compute empirical means for each segment

We first need to calculate the segment means, $\bar{y}_{1:\tau}$ and $\bar{y}_{(\tau+1):n}$, for each τ.

- For $\tau = 1$, the left segment is: $y_{1:1} = (0.5)$, and $\bar{y}_{1:1} = 0.5$. The right segment: $y_{2:4} = (-0.1, 12.1, 12.4)$ gives $\bar{y}_{2:4} = \frac{-0.1+12.1+12.4}{3} = \frac{24.4}{3} = 8.13$.
- For $\tau = 2$, we have, in a similar fashion, $\bar{y}_{1:2} = \frac{0.5-0.1}{2} = 0.2$, $\bar{y}_{3:4} = \frac{12.1+12.4}{2} = 12.25$,
- Lastly, for $\tau = 3$, we have $\bar{y}_{1:3} = \frac{0.5-0.1+12.1}{3} = \frac{12.5}{3} = 4.16$ and $\bar{y}_{4:4} = 12.4$.

Compute the CUSUM statistics

Now that we have the empirical means for each segment, we have all the ingredients for computing our CUSUM:

$$C_\tau = \sqrt{\frac{\tau(n-\tau)}{n}} \left| \bar{y}_{1:\tau} - \bar{y}_{(\tau+1):n} \right|.$$

- **For** $\tau = 1$:
$$C_1 = \sqrt{\frac{1(4-1)}{4}} \left| 0.5 - 8.13\overline{3} \right| = 0.866 \times 7.63\overline{3} = 6.61.$$
- **For** $\tau = 2$:
$$C_2 = \sqrt{\frac{2(4-2)}{4}} \left| 0.2 - 12.25 \right| = 1 \times 12.05 = 12.05.$$
- **For** $\tau = 3$:
$$C_3 = \sqrt{\frac{3(4-3)}{4}} \left| 4.16\overline{6} - 12.4 \right| = 0.866 \times 8.23\overline{3} = 7.13.$$

Thus, the maximum of the CUSUM statistic occurs at $\tau = 2$, with $C_{max} = 12.05$. To detect a changepoint, we would compare C_{max} to a threshold value c. If $C_{max} > c$, we conclude that there is a changepoint at $\hat{\tau} = 2$.

1.3.4 Algorithmic Formulation of the CUSUM Statistic

This process seems rather long, as for every step, we need to precompute the means… A naive implementation of the cusum, in fact, takes $\mathcal{O}(n^2)$ computations.

However, there's an algorithmic trick: by sequentially computing partial sums, e.g. $S_n = \sum_{i=1}^{n} y_i$, we can shorten out our computations significantly. In this way we can compute the value of the means directly as we iterate in the for cycle.

INPUT: Time series $y = (y_1, ..., y_n)$, threshold c, variance σ^2.
OUTPUT: Changepoint estimate $\hat{\tau}$, maximum CUSUM statistic C_{max}

$n \leftarrow$ length of y
$C_{max} \leftarrow 0$
$\hat{\tau} \leftarrow 0$
$S_n \leftarrow \sum_{i=1}^{n} y_i$ // Compute total sum of y
$S \leftarrow 0$

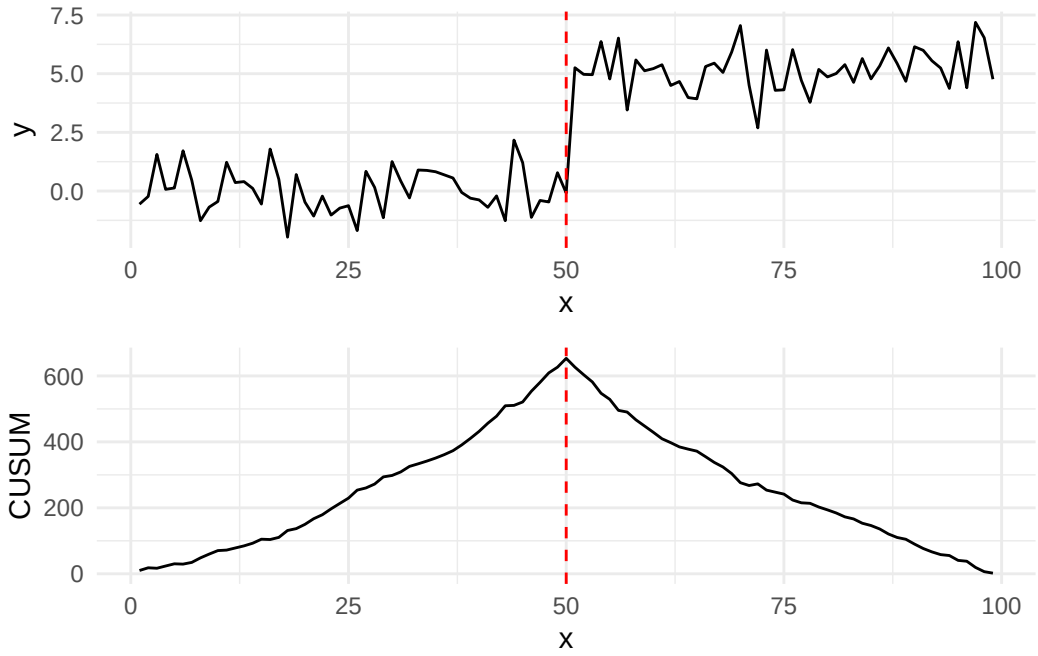
FOR $t = 1, \dots, n-1$
  $S \leftarrow S + y_t$
  $\bar{y}_{1:t} \leftarrow S/t$
  $\bar{y}_{(t+1):n} \leftarrow (S_n - S)/(n-t)$ // Can you figure out why?
  $C_t^2 \leftarrow \frac{t(n-t)}{n}(\bar{y}_{1:t} - \bar{y}_{(t+1):n})^2$
  IF $C_t^2 > C_{max}$
    $C_{max} \leftarrow C_t^2$
    $\hat{\tau} \leftarrow t$

IF $C_{max}/\sigma^2 > c$
  RETURN $\hat{\tau}$, C_{max} // Changepoint detected
ELSE
  RETURN NULL, C_{max} // No changepoint detected

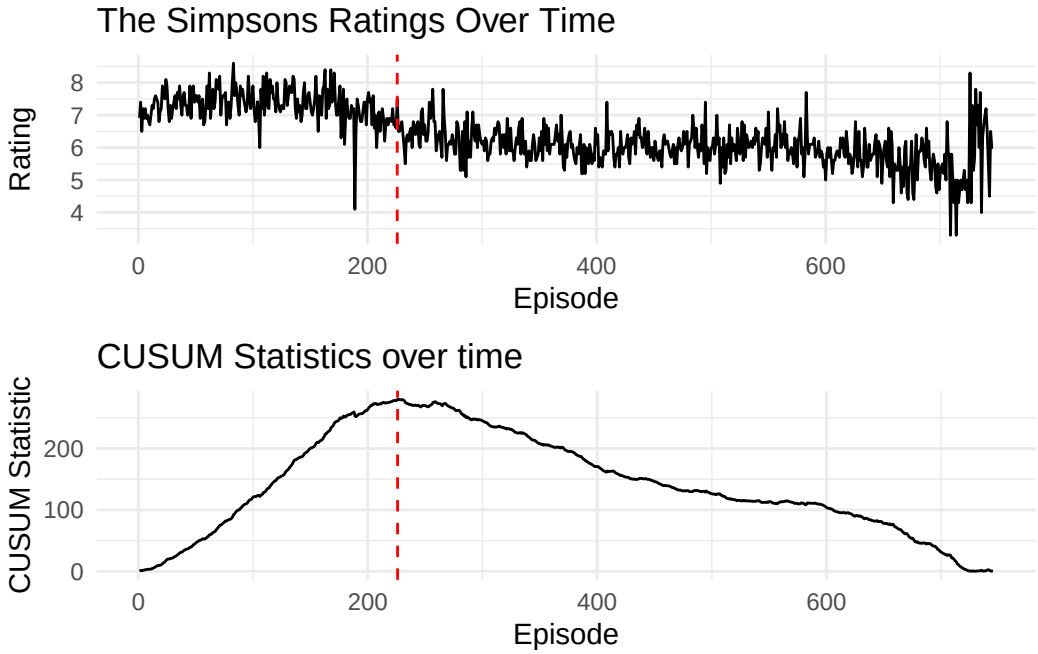
For this reason, the time complexity of the CUSUM algorithm is $O(n)$, where n is the length of the time series.

1.3.5 Example: a large sequence

We can see how the value C_t^2 in the algorithm above behaves across different values of $t = 1, \dots, n-1$ in the example below:



Running the CUSUM test, and maximising on our Simpsons episode, results in:



This results in episode Thirty Minutes over Tokyo being the last "good" Simpsons episode, with Beyond Blunderdome being the start of the decline, according to the Gaussian change-in-mean model!

1.4 Exercises

1.4.1 Code the CUSUM algorithm for a unknown change location, based on the pseudocode above.

Workshop 1

1. Determine if the following processes are stationary, piecewise stationary, or non-stationary:

a. $y_t = y_{t-1} + \epsilon_t, \quad t = 2, \dots, n, y_1 = 0, \epsilon_t \sim N(0,1)$. This is a random walk model. Let's start by computing the expected value and variance of y_t across all t. **TIP:** Start by expanding y_t in terms of the noise components…

b. $y_t = t\epsilon_t + 3\mathbb{1}(t > 50), \quad t = 1, \dots, 100, \quad \epsilon_t \sim N(0,1)$

c. $y_t = 0.05 \cdot t + \epsilon_t, \ t = 1, \dots, 100, \quad \epsilon_t \sim N(0,1)$

2. In this exercise we will show that:

$$\frac{1}{\sigma}\sqrt{\frac{\tau(n-\tau)}{n}}(\bar{y}_{1:\tau} - \bar{y}_{(\tau+1):n})$$

in case of no change, e.g. for $\mu_1 = \mu_2 = \cdots = \mu_n = \mu$, follows a standard normal distribution. **Hint:**

a. Compute the expected value and variance of the difference $\bar{y}_{1:\tau} - \bar{y}_{(\tau+1):n}$

b. Conclude that if you standardise the sum, this follows a standard normal distribution.

1.4.2 Lab 1

1. Code the CUSUM algorithm for a unknown change location, based on the pseudocode of Section Section 1.3.4.
2. Modify your function above to output the CUSUM statistics over all ranges of tau.
3. Recreate the "CUSUM Statistics over time" plot for the Simpsons data above.

a. You'll be able to load the dataset via:

```r
library(tidyverse)
simpsons_episodes <- read_csv("https://www.lancaster.ac.uk/~romano/teaching/2425MATH337/datas

simpsons_ratings <- simpsons_episodes |>
  mutate(Episode = id + 1, Season = as.factor(season), Rating = tmdb_rating)
simpsons_ratings <- simpsons_ratings[-nrow(simpsons_ratings), ]

# run your CUSUM algorithm on the Rating variable!
```

b. To run it on the whole sequence, you'll have to set the threshold $c = \infty$.

c. Assume $\sigma^2 = 1$

2 Controlling the CUSUM and Other Models

In this chapter, we explore the properties of the CUSUM test for detecting a change in mean, and this will allow us how to determine appropriate thresholds, and explore its properties when a changepoint is present.

We will employ some concepts from asymptotic theory: in time series analysis, an asymptotic distribution refers to the distribution that our test statistic approaches as the length of the time series n becomes very large.

2.1 The asymptotic distribution of the CUSUM statistics

If $z_1, \cdots, z_k$ are independent, standard Normal random variables, then:

$$\sum_{i=1}^{k} z_i^2 \sim \chi_k^2,$$

where χ_k^2 is a chi-squared distribution with k degrees of freedom. The chi-squared distribution is a continuous probability distribution that models the sum of squares of k independent standard normal random variables: we have met the chi-squared distribution already in hypothesis testing and constructing confidence intervals. The shape of the distribution depends on its degrees of freedom. For $k = 1$, it's highly skewed, but as k increases, it becomes more symmetric and approaches a normal distribution.

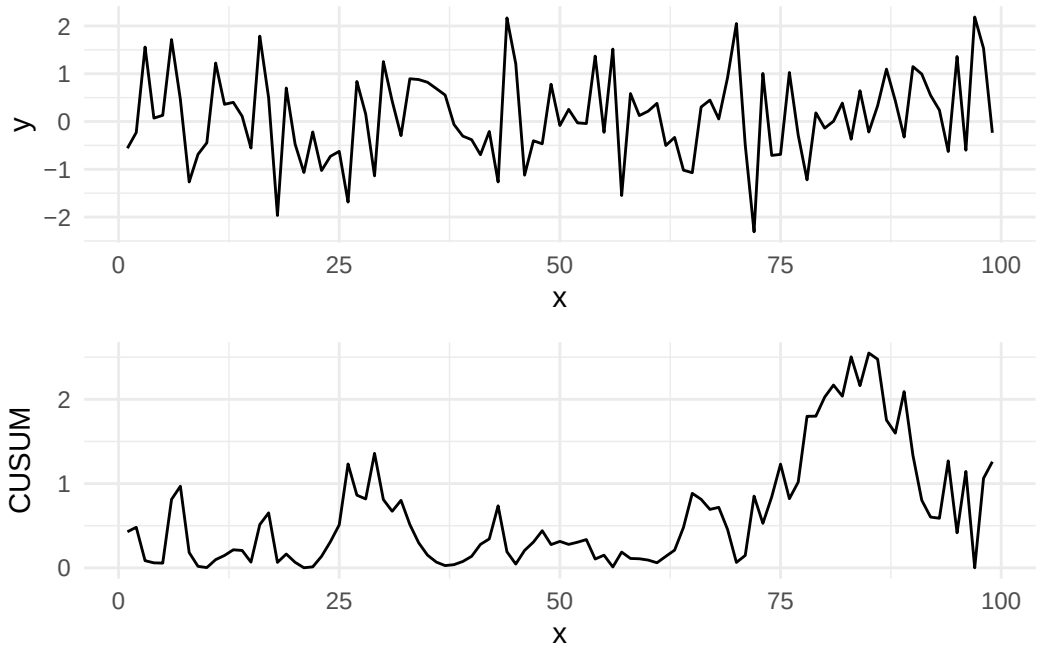
Last week, we found out that, under the null hypothesis of no change:

$$\frac{1}{\sigma}\sqrt{\frac{\tau(n-\tau)}{n}}(\bar{y}_{1:\tau} - \bar{y}_{(\tau+1):n}) \sim N(0,1).$$

Therefore, our test statistics for a fixed τ:

$$\frac{C_\tau^2}{\sigma^2} \sim \chi_1^2.$$

If we take the example of last week, and remove the changepoint, we can observe that the cusum statistics stays constant, and relatively small:



However, as the change is unknown, our actual test statistic for detecting a change is $\max_\tau C_\tau^2/\sigma^2$.

For this reason, calculating the distribution of this maximum ends up being a bit more challenging…

1. So far, we only studied the behaviour of the statistics for one fixed τ, however, when comparing the maximums, the values of C_τ are in fact not independent across different τs.
2. As we will learn later, the CUSUM is a special case of a LR test, as setting the size of the actual change in mean to 0 effectively removes the changepoint parameter from the model. For this reason, the usual regularity conditions for likelihood-ratio test statistics don't apply here.

2.1.1 Controlling the max of our cusums

Fortunately, for controlling our CUSUM test, we can use the fact that $(C_1, ..., C_{n-1})/\sigma$ are the absolute values of a Gaussian process with mean 0 and known covariance, and there are well known statistical results that can help us in our problem. Yao and Davis (1986), in fact, show that the maximum of a set of Gaussian random variables is known to converge to a Gumbel distribution, described by the following equation:

$$\lim_{n\to\infty} \Pr\{a_n^{-1}(\max_\tau C_\tau/\sigma - b_n) \leq u_\alpha\} = \exp\{-(2\pi)^{-1/2}\exp(-u_\alpha)\}, \tag{2.1}$$

where $a_n = (2 \log\log n)^{-1/2}$ and $b_n = a_n^{-1} + 0.5 a_n \log\log\log n$ are a scaling and a centering constant.

The right side of this equation is the CDF of a Gumbell distribution. As we learned from likelihood inference, to find the threshold c_α for a given false probability rate, we first set the right-hand side equal to $1 - \alpha$, and solve for u_α. This gives:

$$u_\alpha = -\log\left(-\frac{\log(1-\alpha)}{(2\pi)^{-1/2}}\right).$$

Then, we can find the critical value by looking into the left side of the equation:

$$\tilde{c} = (a_n u_\alpha + b_n),$$

To find the threshold, as $\max_\tau \frac{C_\tau^2}{\sigma^2} > c$, we just have to square our value above, e.g. $c_\alpha = \tilde{c}^2$.

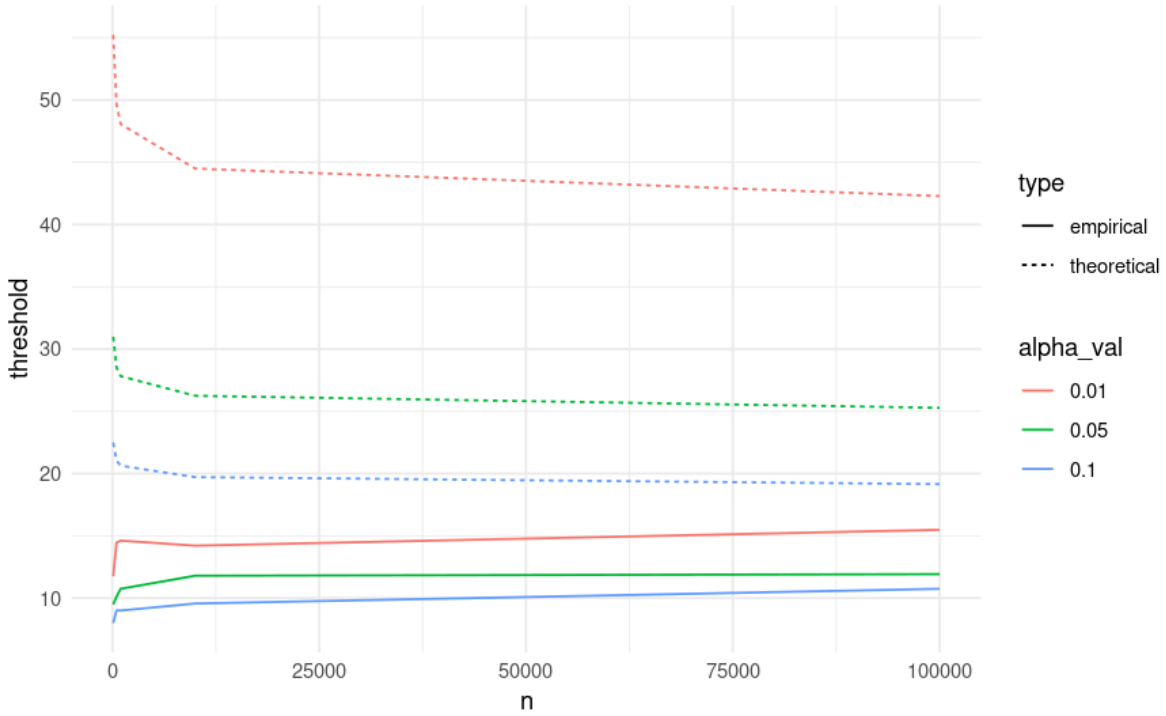
This asymptotic result suggests that the threshold c_α for C_τ^2/σ^2 should increase with n at a rate of approximately $2 \log\log n$. Given that this is a fairly slow rate of convergence, this suggests that the threshold suggested by this asymptotic distribution can be conservative in practice, potentially leading to detect less changepoints than what actually exist.

In practice, it's often simplest and most effective to use Monte Carlo methods to approximate the null distribution of the test statistic. This can be done via the following process:

1. Simulate many time series under the null hypothesis (no changepoint),
2. Calculate the test statistic C_τ^2/σ^2 for each one of the replicates.
3. Set the threshold to be the $(1 - \alpha)$ percentile of the distribution of the test statistics from simulated data.

This leads to have less conservative thresholds.

Theoretical vs Empirical Thresholds The figure below shows, for various levels of $\alpha = 0.01, 0.05, 0.1$, thresholds c_α computed from the theoretical distribution of Equation 2.1 against the Monte Carlo thresholds obtained from empirical simulations under the null.



We will see how to compute in practice the theoretical and empirical thresholds in the Lab!

2.2 The Likelihood Ratio Test

The CUSUM can be viewed as a special case of a more general framework based on the Likelihood Ratio Test (LRT). This allow us to test for more general settings, beyond simply detecting changes in the mean.

In general, the Likelihood Ratio Test is a method for comparing two nested models: one under the null hypothesis, which assumes no changepoint, and one under the alternative hypothesis, which assumes a changepoint exists at some unknown position τ.

Suppose we have a set of observations $y_1, y_2, \dots, y_n$. Under the null hypothesis H_0, we assume that all the data is generated by the same model without a changepoint. Under the alternative hypothesis H_1, there is a single changepoint at τ, such that the model for the data changes after τ. The LRT statistic is given by:

$$LR_\tau = -2 \log \left\{ \frac{\max_\theta \prod_{t=1}^{n} f(y_t|\theta)}{\max_{\theta_1,\theta_2} [(\prod_{t=1}^{n} f(y_t|\theta_1))(\prod_{t=1}^{n} f(y_t|\theta_2)]} \right\} \tag{2.2}$$

The LRT compares the likelihood of the data under two models to determine which one is more likely: the enumerator, is the likelihood under the null hypothesis of no changepoint, while the denominator represents the likelihood of the data under the alternative hypothesis, where we optimise for two different parameters before and after the changepoint at τ.

2.2.1 Example: Gaussian change-in-mean

As a first example, we show how the CUSUM statistics is nothing but a specific case of the GLR. To see this, we start from our piecewise costant signal, plus noise, $y_t = f_t + \epsilon_t, \quad t = 1, \dots, n$. Under this model our data, a linear combination of a Gaussian, is distributed as:

$$y_t \sim N(\mu_t, \sigma^2), \quad t = 1, \dots, n$$

Our p.d.f. will be:

$$f(y_t|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(y_t - \mu)^2\}.$$

Therefore, to obtain the likelihood ratio test statistic, we plug our Gaussian p.d.f. into the LR above, and take the logarithm:

$$LR_\tau = -2 \left[\max_{\mu} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu)^2 \right) - \max_{\mu_1, \mu_2} \left(-\frac{1}{2\sigma^2} \left(\sum_{i=1}^{\tau} (y_i - \mu_1)^2 + \sum_{i=\tau+1}^{n} (y_i - \mu_2)^2 \right) \right) \right] + \tag{2.3}$$

$$+ \tau \log(2\pi\sigma^2) + (n - \tau) \log(2\pi\sigma^2) - n \log(2\pi\sigma^2). \tag{2.4}$$

This simplifies to:

$$= \frac{1}{\sigma^2} \left[\min_{\mu} \sum_{i=1}^{n} (y_i - \mu_1)^2 - \min_{\mu_1, \mu_2} \left(\sum_{i=1}^{\tau} (y_i - \mu_1)^2 + \sum_{i=\tau+1}^{n} (y_i - \mu_2)^2 \right) \right].$$

To solve the minimization over μ_1 and μ_2, we plug-in values $\hat{\mu} = \bar{y}_{1:n}$ on the first term, and $\hat{\mu}_1 = \bar{y}_{1:\tau}$, $\hat{\mu}_2 = \bar{y}_{(\tau+1):n}$ for the second term:

$$LR_\tau = \frac{1}{\sigma^2} \left[\sum_{i=1}^{n} (y_i - \bar{y}_{1:n})^2 - \sum_{i=1}^{\tau} (y_i - \bar{y}_{1:\tau})^2 - \sum_{i=\tau+1}^{n} (y_i - \bar{y}_{(\tau+1):n})^2 \right].$$

This is the likelihood ratio test statistic for a change in mean in a Gaussian model, which is essentially the CUSUM statistics squared, rescaled by the known variance:

$$LR_\tau = \frac{C_\tau^2}{\sigma^2}.$$

It is possible to prove this directly with some tedious computations.

Proof. We start by writing down $\sigma^2 LR$. This will be:

$$\sigma^2 LR_\tau = \sum_{i=1}^{n}(y_i - \bar{y}_{1:n})^2 - \sum_{i=1}^{\tau}(y_i - \bar{y}_{1:\tau})^2 - \sum_{i=\tau+1}^{n}(y_i - \bar{y}_{\tau+1:n})^2.$$

Now we need to expand each term. Starting with the first:

$$\sum_{i=1}^{n}(y_i - \bar{y}_{1:n})^2 = \sum_{i=1}^{n} y_i^2 - 2\bar{y}_{1:n}\sum_{i=1}^{n} y_i + n\bar{y}_{1:n}^2.$$

As $\sum_{i=1}^{n} y_i = n\bar{y}_{1:n}$, we notice that we can simplify the last two terms. We are left with:

$$\sum_{i=1}^{n}(y_i - \bar{y}_{1:n})^2 = \sum_{i=1}^{n} y_i^2 - n\bar{y}_{1:\tau}^2.$$

We proceed similarly for the other two terms:

$$\sum_{i=1}^{\tau}(y_i - \bar{y}_{1:\tau})^2 = \sum_{i=1}^{\tau} y_i^2 - \tau\bar{y}_{1:\tau}^2, \quad \sum_{i=\tau+1}^{n}(y_i - \bar{y}_{\tau+1:n})^2 = \sum_{i=\tau+1}^{n} y_i^2 - (n-\tau)\bar{y}_{\tau+1:n}^2.$$

Putting all together, and getting rid of the partial sums, we are left with:

$$\sigma^2 LR_\tau = -n\bar{y}_{1:n}^2 + \tau\bar{y}_{1:\tau}^2 + (n-\tau)\bar{y}_{\tau+1:n}^2.$$

Now, recall that $\bar{y}_{1:n} = \frac{1}{n}\left[\tau\bar{y}_{1:\tau} + (n-\tau)\bar{y}_{\tau+1:n}\right]$, and:

$$\bar{y}_{1:n}^2 = \frac{1}{n^2}\left[\tau^2\bar{y}_{1:\tau}^2 + 2\tau(n-\tau)\bar{y}_{1:\tau}\bar{y}_{\tau+1:n} + (n-\tau)^2\bar{y}_{\tau+1:n}^2\right].$$

Plugging in this into our LR:

$$\sigma^2 LR_\tau = -\frac{\tau^2}{n}\bar{y}_{1:\tau}^2 - \frac{2\tau(n-\tau)}{n}\bar{y}_{1:\tau}\bar{y}_{\tau+1:n} - \frac{(n-\tau)^2}{n}\bar{y}_{\tau+1:n}^2 - \tau\bar{y}_{1:\tau}^2 - (n-\tau)\bar{y}_{\tau+1:n}^2 = \quad (2.5)$$

$$= \frac{\tau(n-\tau)}{n}\bar{y}_{1:\tau}^2 - \frac{2\tau(n-\tau)}{n}\bar{y}_{1:\tau}\bar{y}_{\tau+1:n} + \frac{\tau(n-\tau)}{n}\bar{y}_{\tau+1:n}^2 = \quad (2.6)$$

$$= \frac{\tau(n-\tau)}{n}(\bar{y}_{1:\tau}^2 - 2\bar{y}_{1:\tau}\bar{y}_{\tau+1:n} + \bar{y}_{\tau+1:n}^2) = \quad (2.7)$$

$$= \frac{\tau(n-\tau)}{n}(\bar{y}_{1:\tau} - \bar{y}_{\tau+1:n})^2 = \quad (2.8)$$

$$= C_\tau^2. \quad (2.9)$$

This gives us $LR_\tau = \frac{C_\tau^2}{\sigma^2}$.

2.3 Towards More General Models

The great thing of the LR test is that it's extremely flexible, allowing us to detect other changes then the simple change-in-mean case. As before, the procedure is to compute the LR test conditional on a fixed location of a changepoint, e.g. LR_τ, and range across all possible values for τ to find the test statistics for our change.

2.3.1 Change-in-variance

To this end we will demonstrate how to construct a test for Gaussian change-in-variance, for mean known. For simplicity, we will call our variance $\sigma^2 = \theta$, our parameter of interest, and without loss of generality, we can center our data on zero (e.g. if $x_t \sim N(\mu, \theta)$, then $x_t - \mu = y_t \sim N(0, \theta)$). Then, our p.d.f for one observation will be given by:

$$f(y_t|\theta) = \frac{1}{\sqrt{2\pi\theta}}\exp\{-\frac{y_t^2}{2\theta}\}.$$

Plugging in the main LR test formula, we find:

$$LR_\tau = -2\log\left\{\frac{\max_\theta \prod_{t=1}^n \frac{1}{\sqrt{2\pi\theta}}\exp\{-\frac{y_t^2}{2\theta}\}}{\max_{\theta_1,\theta_2}[(\prod_{t=1}^\tau \frac{1}{\sqrt{2\pi\theta_1}}\exp\{-\frac{y_t^2}{2\theta_1}\})(\prod_{t=\tau+1}^n \frac{1}{\sqrt{2\pi\theta_2}}\exp\{-\frac{y_t^2}{2\theta_2}\}]}\right\}$$

And taking the log, and simplifying over the constant gives us:

$$LR_\tau = -\max_\theta \sum_{t=1}^{n} \left(-\log(\theta) - \frac{y^2}{\theta} \right) + \max_{\theta_1,\theta_2} \left[\sum_{t=1}^{\tau} \left(-\log(\theta_1) - \frac{y^2}{\theta_1} \right) + \sum_{t=\tau+1}^{n} \left(-\log(\theta_2) - \frac{y^2}{\theta_2} \right) \right] = \tag{2.10}$$

$$= \min_\theta \sum_{t=1}^{n} \left(\log(\theta) + \frac{y^2}{\theta} \right) - \min_{\theta_1,\theta_2} \left[\sum_{t=1}^{\tau} \left(\log(\theta_1) + \frac{y^2}{\theta_1} \right) + \sum_{t=\tau+1}^{n} \left(\log(\theta_2) + \frac{y^2}{\theta_2} \right) \right] \tag{2.11}$$

Now to solve the minimisation, we focus on the first term:

$$f(y_{1:n}, \theta) = \sum_{t=1}^{n} \left(\log(\theta) + \frac{y^2}{\theta} \right) = \left(n \log(\theta) + \frac{\sum_{t=1}^{n} y^2}{\theta} \right).$$

Taking the derivative with respect to θ, gives:

$$\frac{d}{d\theta} f(y_{1:n}, \theta) = \frac{n}{\theta} - \frac{\sum_{t=1}^{n} y^2}{\theta^2}.$$

Setting equal to zero and solving for θ:

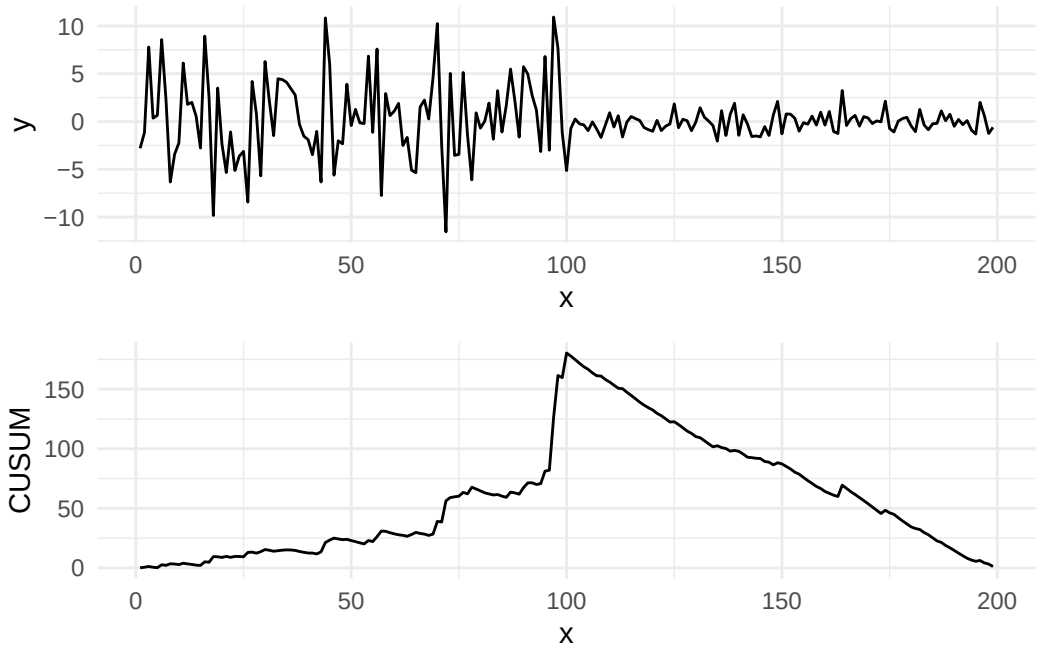
$$n\theta - \sum_{t=1}^{n} y^2 = 0$$

Which gives us: $\hat{\theta} = \frac{\sum_{t=1}^{n} y^2}{n} = \bar{S}_{1:n}$ the sample variance.

Solving the optimization for θ_1 and θ_2 similarly, gives us the values $\hat{\theta}_1 = \bar{S}_{1:\tau},\ \hat{\theta}_2 = \bar{S}_{(\tau+1):n}$.

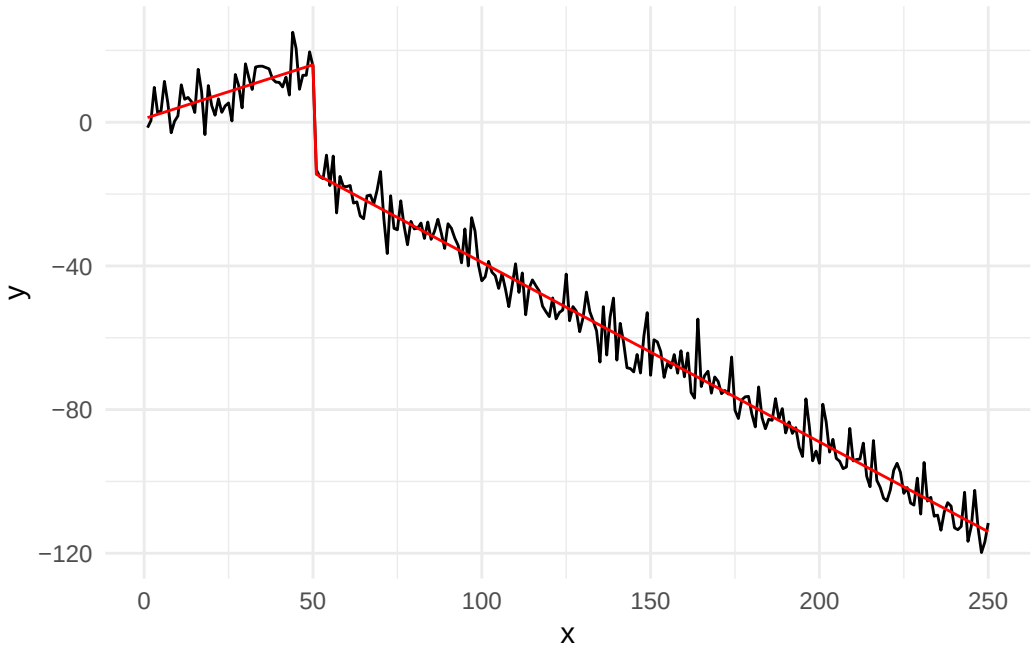
Now, as $f(y_{1:n}, \hat{\theta}) = n \log(\bar{S}_{1:n}) + n$ (why?) the final LR test simplifies to:

$$LR_\tau = \left[n \log(\bar{S}_{1:n}) - \tau \log(\bar{S}_{1:\tau}) - (n - \tau) \log(\bar{S}_{(\tau+1):n}) \right]$$



2.3.2 Change-in-slope

Another important example, and an alternative to detecting a change-in-mean, is detecting a change in slope. In this section, we assume the data is still modeled as a signal plus noise, but the signal itself is a linear function of time (e.g. non-stationary, with a change!). Graphically:



More formally, let our data be modeled as:

$$y_t = f_t + \epsilon_t,\ \epsilon_t \sim N(0,1) \quad t = 1, \dots, n.$$

In this scenario, for simplicity, we assume a known constant variance, which without loss of generality, we take to be 1.

Under the null hypothesis H_0, we assume that the signal is linear with a constant slope over the entire sequence, i.e.,

$$f_t = \alpha_1 + t\theta_1, \quad t = 1, \dots, n,$$

where α_1 is the intercept, and θ_1 is the slope. However, under the alternative hypothesis H_1, we assume there is a changepoint at τ after which the slope changes. Thus, the signal becomes:

$$f_t = \alpha_1 + t\theta_1, \quad t = 1, \dots, \tau; \quad f_t = \alpha_2 + t\theta_2, \quad t = \tau + 1, \dots, n,$$

where α_2 is the new intercept, and θ_2 is the new slope after the changepoint. In other words, the model is showing a piecewise linear mean.

For this model, the log-likelihood ratio test statistic can be written as the square of a projection of the data onto a vector v_τ, i.e.,

$$LR_\tau = \left(v_\tau^\top y_{1:n}\right)^2,$$

where v_τ is a contrast vector that is piecewise linear with a change in slope at τ. This vector is constructed such that, under the null hypothesis, the vector $v_\tau^\top y_{1:n}$ has variance 1, and $v_\tau \top y_{1:n}$ is invariant to adding a linear function to the data. These properties uniquely define the contrast vector v_τ, up to an arbitrary sign. Computations on how to obtain this likelihood ration test, and how to construct this vector are beyond the scope of this module, but should you be curious those are detailed in Baranowski, Chen, and Fryzlewicz (2019).

2.3.3 Revisiting our Simpsons data (again!)

So, going back to the Simpsons example... We mentioned how the belowed show rose rapidly to success, and at one point, started to *decline*... A much better model would therefore be our change-in-slope model!

To run the model, we can take advantage of the `changepoint` package, which by default is a multiple changepoint package (we will see these in the next week), but whose simplest case implements exactly our change-in-slope LR test.

Before we proceed, we need to load, clean and standardize our data:

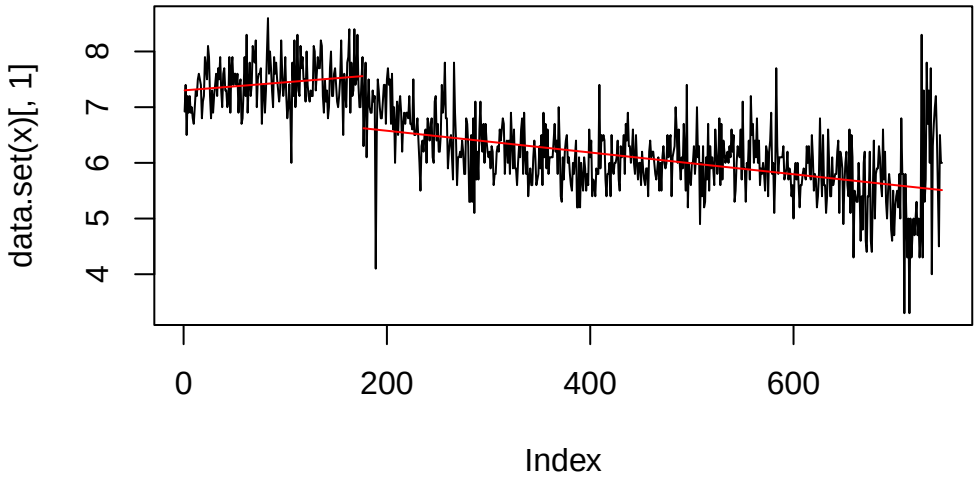
```
# Load Simpsons ratings data
simpsons_episodes <- read.csv("extra/simpsons_episodes.csv")
simpsons_episodes <- simpsons_episodes |>
  mutate(Episode = id + 1, Season = as.factor(season), Rating = tmdb_rating)
simpsons_episodes <- simpsons_episodes[-nrow(simpsons_episodes), ]

y <- simpsons_episodes$Rating
```

We can then run our model with:

```
library(changepoint)

data <- cbind(y, 1, 1:length(y))

out <- cpt.reg(data, method="AMOC") # AMOC is short for "At Most One Change"

print(paste0("Our changepoint estimate (chagepoints): ",  cpts(out)))
```

```
[1] "Our changepoint estimate (chagepoints): 176"
```

```r
plot(out)
```



We can see that we now find a significant changepoint prior to episode The Simpsons Spin-Off Showcase, which is anthology episode well over into season 8, which, according to our method, is the beginning of the decline! However, some among you, might have noticed that there are more then one changes in this dataset… We will see, in fact, how we can improve on our estimation in the following weeks!

2.4 Exercises

2.4.1 Workshop 2

1. Compute the LR ratio to detect a change in the success probability of a Bernoulli Random Variable.

    a. Start by writing down the distribution of the model under the null, and find the MLE. Extend this to the alternative

    b. Compose the log-likelihood ratio, according to the equation Equation 2.2 introduced above.

2.4.2 Lab 2

1. Write a function, that taking as input n and a desired α level for false positive rate, returns the threshold for the cusum statistics, according to Section Section 2.1.1.

2. Construct a function that, taking as input n, a desired α , and a `replicates` parameter, runs a Monte Carlo simulation to tune an empirical penalty for the CUSUM change-in-mean on a simple Gaussian signal. Tip: You can reuse the function for computing the CUSUM statistics that you built the last week

3. Compare for a range of increasingly values of n, e.g. $n = 100, 500, 1000, 10.000$, and for few desired levels of alpha, the Monte Carlo threshold with the theoretically justified threshold. Plot the results, to recreate the plot above.

4. Using the Test the Simpsons dataset, and the monte carlo threshold, find a critical level for your CUSUM statistics, and declare a change with the change-in-mean model.

3 Multiple changepoints

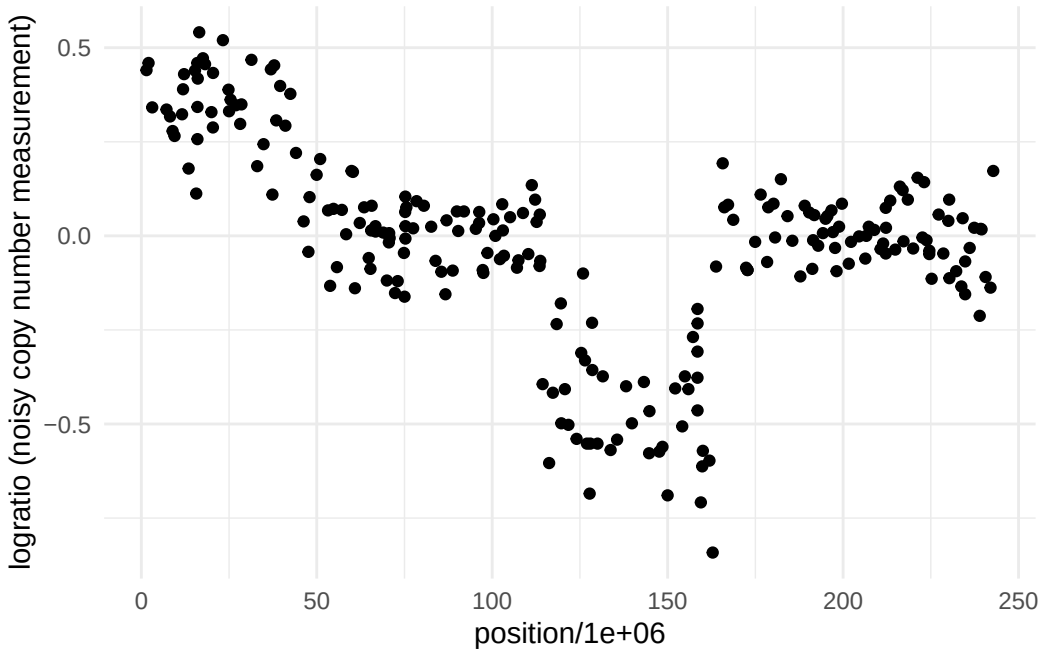
3.1 Introduction

In real-world data, it is common to encounter situations where more than one change occurs. When applying the CUSUM statistic in such cases, where there are **multiple changes**, the question arises: how does CUSUM behave, and how can we detect these multiple changes effectively?

3.1.1 Real Example: Genomic Data and Neuroblastoma

To motivate this discussion, we return to the example from week 1: detecting active genomic regions using *ChIP-seq data*. Our goal here is to identify copy number variations (CNVs)—structural changes in the genome where DNA sections are duplicated or deleted. These variations can impact gene expression and are linked to diseases like cancer, including neuroblastoma. The dataset we'll examine consists of logratios of genomic probe intensities, which help us detect changes in the underlying DNA structure.

Statistically, our objective is to segment this logratio sequence into regions with different means, corresponding to different genomic states:



As seen from the plot, the data is noisy, but there are visible shifts in the logratio values, suggesting multiple changes in the underlying copy number. By the end of this chapter, we will segment this sequence!

3.1.2 Towards multiple changes

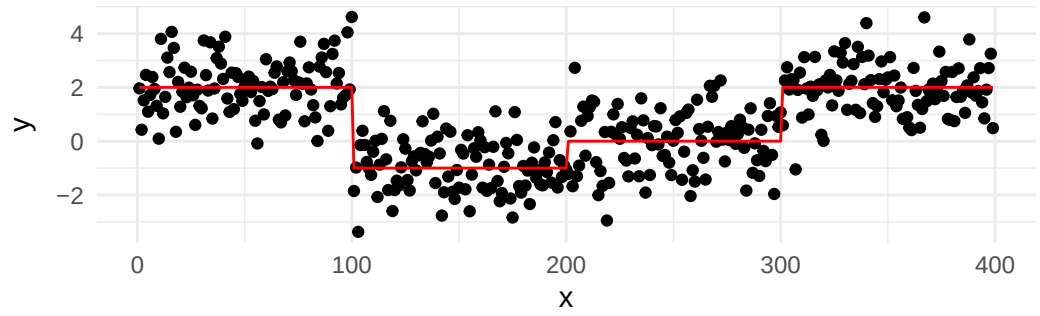
Under this framework, the observed sequence y_t can be modeled as a piecewise constant signal with changes in the parameter, occurring at each changepoint $\tau_1, \dots, \tau_k, \dots, \tau_K$, where $\tau_k < \tau_{k+1}$, with K being the maximum number of changes that we have in the sequence, corresponding to $K+1$ segments. Also - for the sake of notation, in the rest of this module we will set $\tau_0 = 0$ and $\tau_{K+1} = n$.

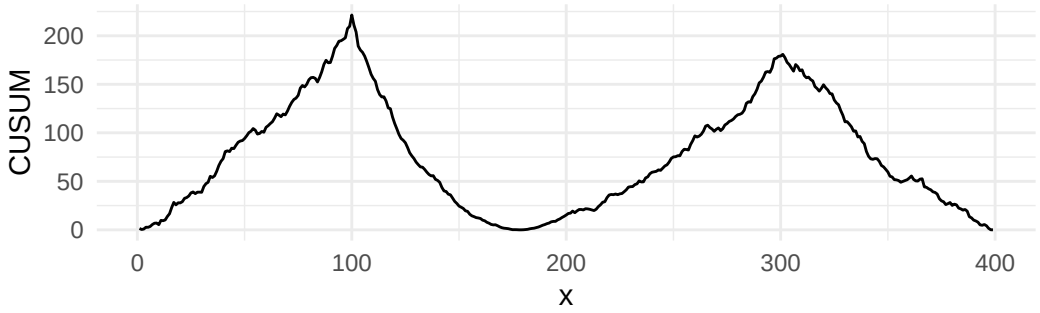
Multiple Changes-in-Mean. A plausible model for the change-in-mean signal is given by

$$y_t = \mu_k + \epsilon_t, \quad \text{for } \tau_k + 1 \leq t < \tau_{k+1},\ k = 0, 1, \dots, K,$$

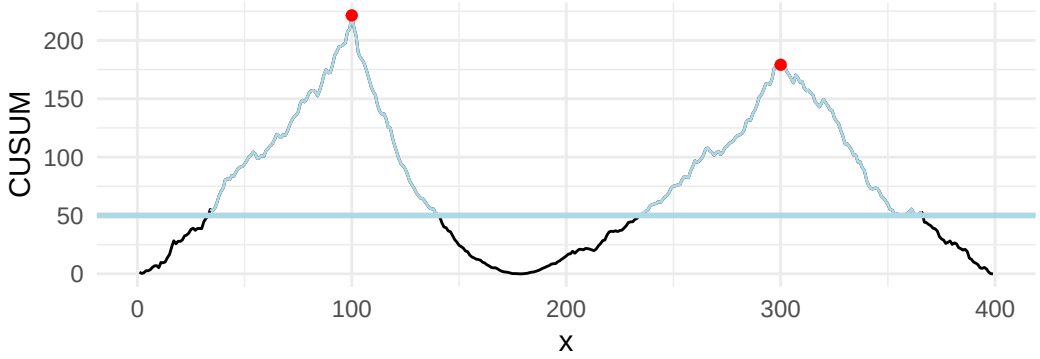
where μ_k is the mean of the k-th segment, and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian noise terms with mean 0 and (known) variance σ^2.

As a starting example, we can generate a sequence with 4 segments, with $\tau_1 = 50, \tau_2 = 100, \tau_3 = 150$ and means $\mu_1 = 2, \mu_2 = 0, \mu_3 = -1$ and $\mu_4 = 2$. Running the CUSUM statistic in this scenario with multiple changes, leads to the following C_τ^2 trace:





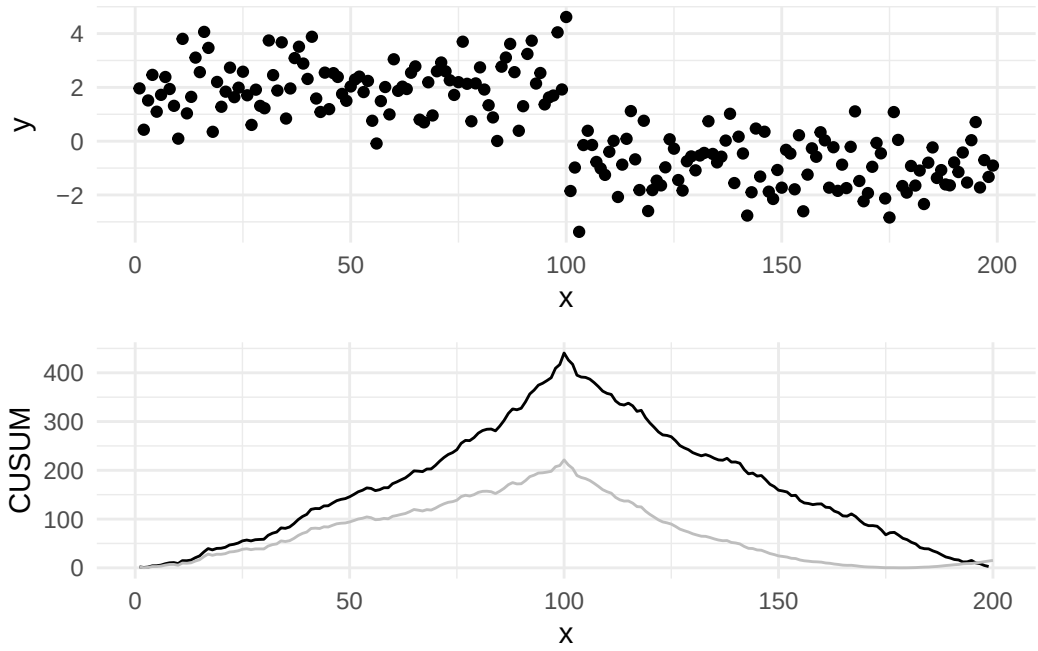
One thing we could do, would be to set a threshold, say 50, record the windows over which the CUSUM is over the threshold, and pick the argmax in each of our windows as the candidate changepoints.



However, from this simple example we notice that we miss already one changepoint, that of time 200… In fact, in some scenarios, such as this one, detection power of the CUSUM statistic lost when there is more then one change in our test.

To make it even worse, if we compare the values of the CUSUM statistic ran on the whole dataset (as above), against with the values of the CUSUM, ran on a subset limited to only only the first two segments $y_{1:200}$:

```
Warning: Removed 199 rows containing missing values or values outside the scale range
(`geom_line()`).
```



We can see that max of the CUSUM across the entire dataset (the line in grey, that we computed before) is much lower than the one where we isolate the sequence on one single change!

So there is an effective loss of power in this scenario when analyzing all changes together, as some segments means are masking the effects of others with the CUSUM…

This gives us motivation to move towards some strategy that tries to estimate all changes locations jointly, rather then looking for one!

3.1.3 The cost of a segmentation

Well, so far we only worked with one scheme that tried to split a sequence in a hald

But how can we work in case we have more than one change? Well, we need to introduce the cost of a segment.

3.1.3.1 The cost of a segment

If we assume the data is independent and identically distributed within each segment, for segment parameter θ, then this cost can be obtained through:

$$\mathcal{L}(y_{s+1:t}) = \min_{\theta} \sum_{i=s+1}^{t} -2\log(f(y_i, \theta)) \tag{3.1}$$

with $f(y, \theta)$ being the likelihood for data point y if the segment parameter is θ. Note, as the parameter of interest is θ, we can remove all constant terms with respect to θ, as those will not affect our optimization.

Example. Now, for example, in the Gaussian case, recall our p.d.f. is given by:

$$f(y_i, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(y_i - \mu)^2\}.$$

Taking the log and summing across all data points in the segment:

$$\sum_{i=s+1}^{t} -2 \log f(y_i|\theta) = -2 \left[-\frac{t-s+1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=s+1}^{t} (y_t - \mu)^2 \right].$$

This is minimized for $\bar{y}_{s+1:t} = \frac{1}{t-s} \sum_{i=s+1}^{t} y_t$. Therefore, plugging this into our equation Equation 3.1, the cost of a segment will be given by:

$$\mathcal{L}(y_{s+1:t}) = (t-s+1) \log(2\pi\sigma^2) + \frac{1}{\sigma^2} \sum_{i=s}^{t} \left(y_i - \bar{y}_{s+1:t}\right)^2.$$

Remember, we can get rid of all constants terms as those do not contribute to our optimization. Doing so, our cost will be simply:

$$\mathcal{L}(y_{s+1:t}) = \frac{1}{\sigma^2} \sum_{i=s}^{t} \left(y_i - \bar{y}_{s+1:t}\right)^2.$$

3.1.3.2 Obtaining the cost of the full segmentation

The cost for the full segmentation will be given by the sum across all segments:

$$\sum_{k=0}^{K} \mathcal{L}(y_{\tau_k+1:\tau_{k+1}})$$

Interestingly, the cost of a full segmentation is closely related to the LR test. Consider, a single Gaussian change-in-mean, e.g. $K = 1$ at time $\tau_1 = \tau$, splitting the data into two segments: $y_{1:\tau}$ and $y_{\tau+1:n}$. The cost of this segmentation is:

$$\mathcal{L}(y_{1:\tau}) + \mathcal{L}(y_{\tau+1:n}) = \frac{1}{\sigma^2} \left[\sum_{i=1}^{\tau} (y_i - \bar{y}_{1:\tau})^2 + \sum_{i=\tau+1}^{n} (y_i - \bar{y}_{(\tau+1):n})^2 \right]$$

Which is essentially minus the LR test as we saw last week, without the null component. Specifically, for one change, minimizing the segmentation cost over all possible changepoints locations τ is equivalent to maximizing the CUSUM statistic.

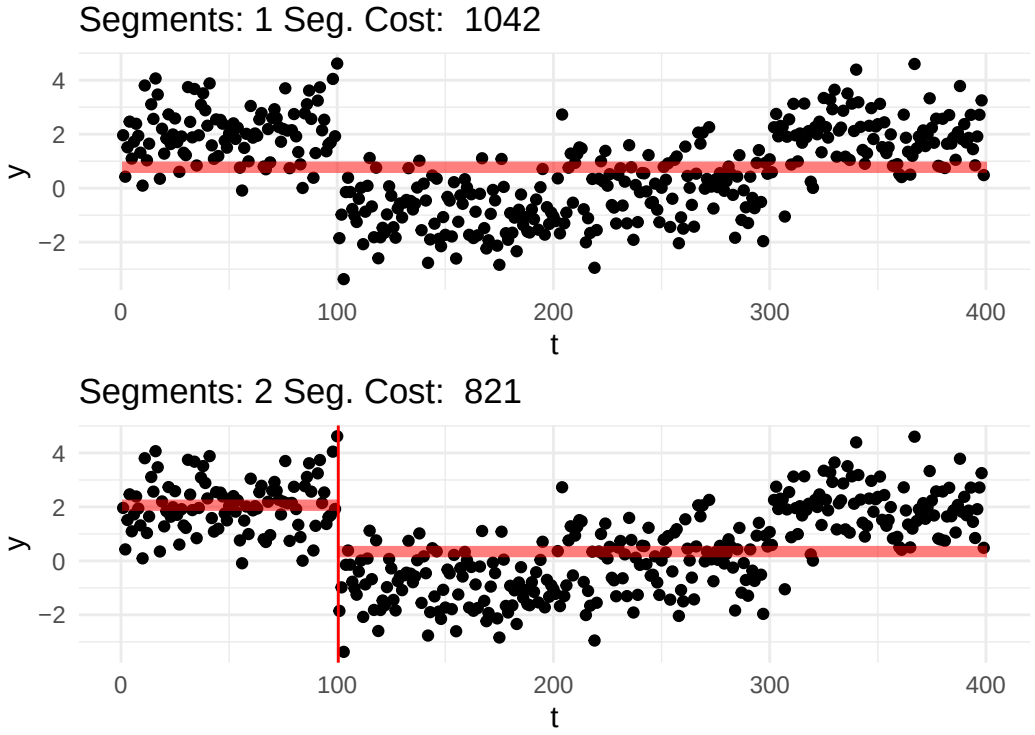
3.1.4 The "best" segmentation

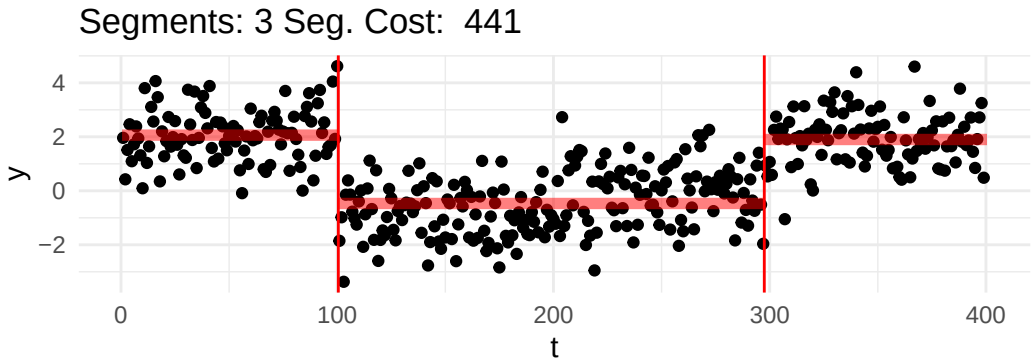
We now have a way of evaluating how "good" a segmentation is, so it's only natural to ask the question: what would be the best one?

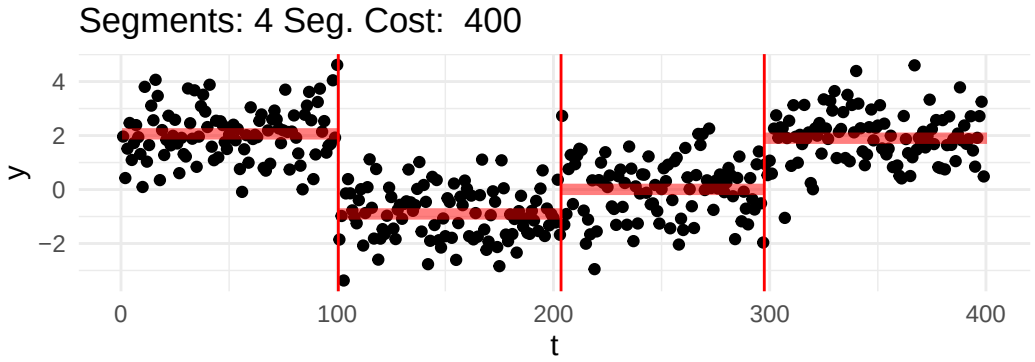
Well, one way would be to, say, finding the the best set of $\tau = \tau_0, \dots, \tau_{K+1}$ changepoints that minimise the cost:

$$\min_{\substack{K \in \mathbb{N} \\ \tau_1, \dots, \tau_K}} \sum_{k=0}^{K} \mathcal{L}(y_{\tau_k+1:\tau_{k+1}}).$$

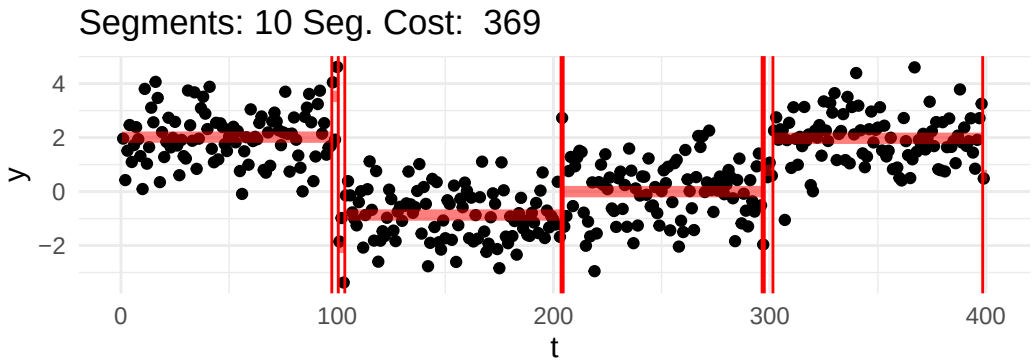
Which one would this be? Say that for instance we range the $K = 1, \dots, n$, and at each step we find the best possible segmentation. Graphically, we would be observing the following:

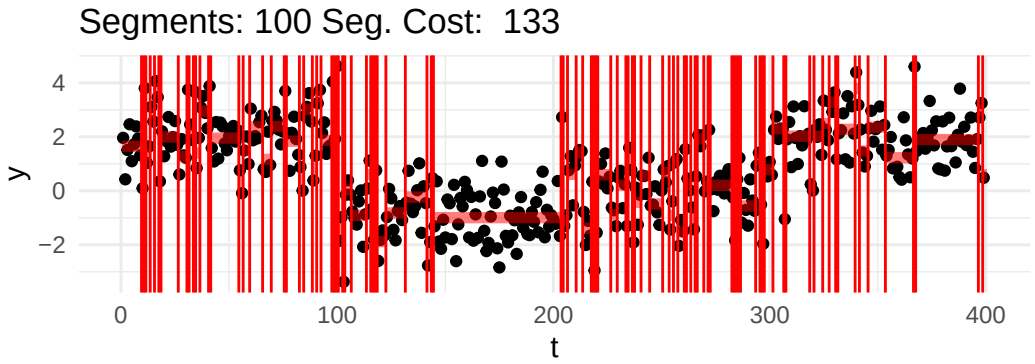




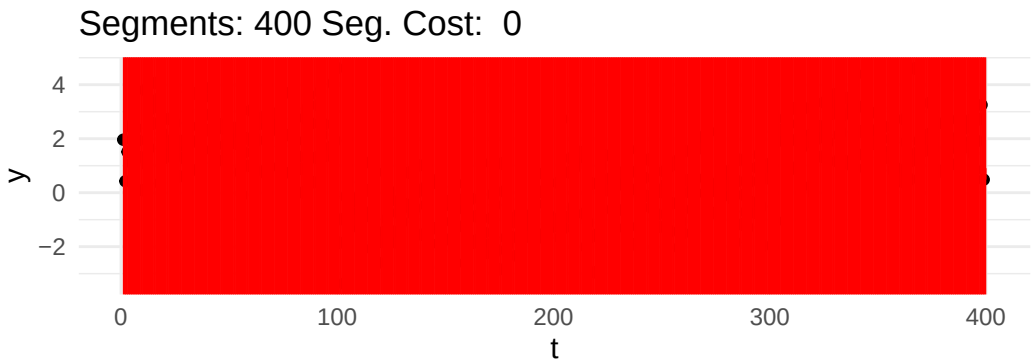


Well, arguably we would like to stop at 4, which we know is the real number of segments, but the cost keep going down...




Segments: 100 Seg. Cost: 133
y
4
2
0
−2
0
100
200
300
400
t

And finally:


Segments: 400 Seg. Cost: 0
y
4
2
0
−2
0
100
200
300
400
t


THIS IS
FINE

Well, it turns out, that according to the minimization above, the optimal segmentation across all would be the one that puts each point into its own segment!

Well, there are different solutions to this problem. The first one we will see, is a divide-and-conquer greedy approach, called Binary Segmentation, and the second one will aim a generating a different optimization to the one below that will find the optimal segmentation *up to a constant* to avoid over-fitting!

3.2 Binary Segmentation

Binary Segmentation (BS) is a procedure from Scott and Knott (1974) and Sen and Srivastava (1975). Binary segmentation works like this:

1. Start with a test for a change τ that splits a sequence into two segments and to check if the cost over those two segments, plus a penalty $\beta \in \mathbb{R}$, is smaller then the cost computed on the whole sequence:

$$\mathcal{L}(y_{1:\tau}) + \mathcal{L}(y_{\tau+1:n}) + \beta < \mathcal{L}(y_{1:n}) \tag{3.2}$$

where the segment cost $\mathcal{L}(\cdot)$, is as in Equation 3.1.

2. If the condition in Equation 3.2 is true for at least one $\tau \in 1, \dots, n$, then the τ that minimizes $\mathcal{L}(y_{1:\tau}) + \mathcal{L}(y_{\tau+1:n})$ is picked as a first changepoint and the test is then performed on the two newly generated splits. This step is repeated until no further changepoints are detected on all resulting segments.

3. If there are no more resulting valid splits, then the procedure ends.

Some of you might have noted how the condition in Equation 3.2 is closely related to the LR test in Equation 2.2. In fact, rearranging equation above, gives us:

$$-\mathcal{L}(y_{1:n}) + \mathcal{L}(y_{1:\tau}) + \mathcal{L}(y_{\tau+1:n}) = -\frac{LR_\tau}{2} < -\beta.$$

The $-\beta$ acts exactly as the constant c for declaring a change, and it adds a natural stopping condition, solving the issue of overfitting that we mentioned in the previous section! Binary Segmentation, in fact, does nothing more then iteratively running a LR test, until no changes are found anymore!

This gives us a strategy to essentially apply a test that is locally optimal for one change, such as the Likelihood Ratio test, to solve a multiple changepoint segmentation. For this reason, BS is often employed to extend single changepoint procedures to multiple changes procedures, and hence it is one of the most prominent methods in the literature.

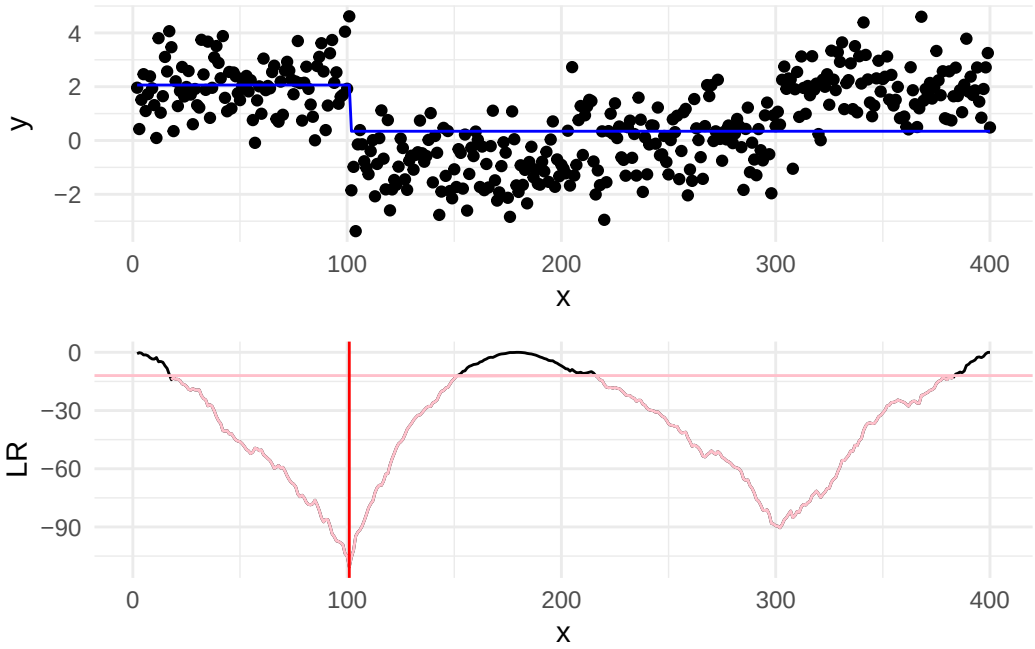
3.2.1 Binary Segmentation in action

Having introduced the main idea, we show now how binary segmentation works in action with an example above. Say that we set a $\beta = 2\log(400) = 11.98$.

Step 1: We start by computing the cost as in Equation 3.2, and for those that are less then β, we pick the smallest. This will be our first changepoint estimate, and the first point of split.

In the plots below, the blue horizontal line is the mean signal estimated for a given split, while in the cusum the pink will represent the values of the LR below the threshold β, and red vertical line will show the min of the test statistics. When the cost is below the beta line, this will be our changepoint estimate.

In our case, we can see that the min of our cost has been achieved for $\hat{\tau} = 100$, and since this is below the threshold, it's our first estimated changepoint!

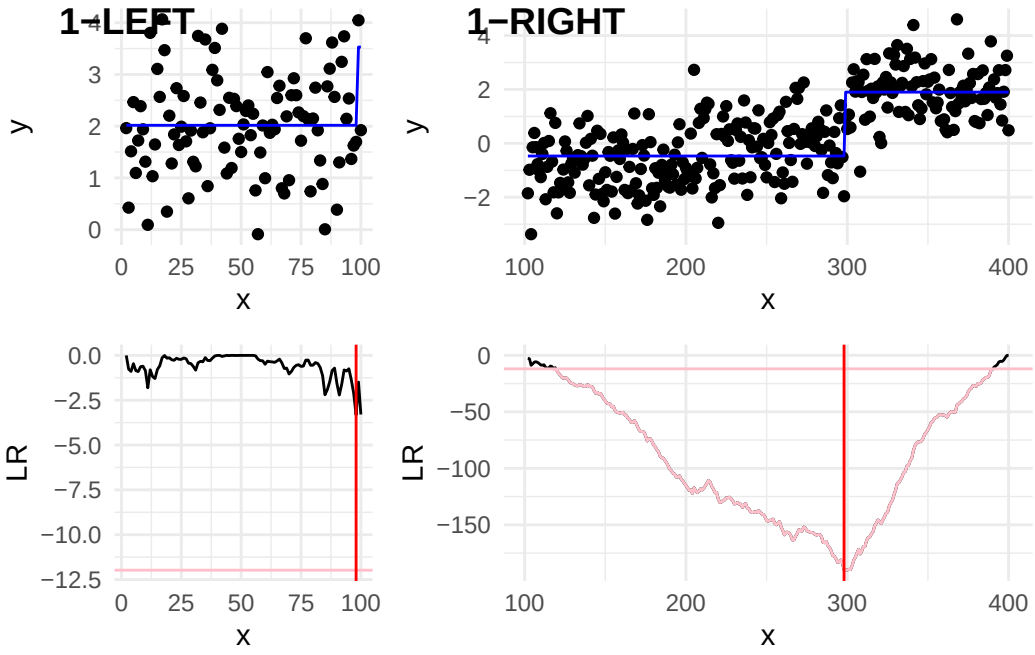


Step 2:

From the first step, we have to check now two splits:

- The first left split, **1-LEFT** in the plot below, covers data $y_{1:100}$. We can see that from here, the min of our statistic is below the threshold, therefore we won't declare any further change in this subset.
- The first right split, **1-RIGHT** covers data $y_{101:400}$. We can see that here, the min of the statistics, is below the threshold, and therefore we identify a second change at $\hat{\tau} = 297$.

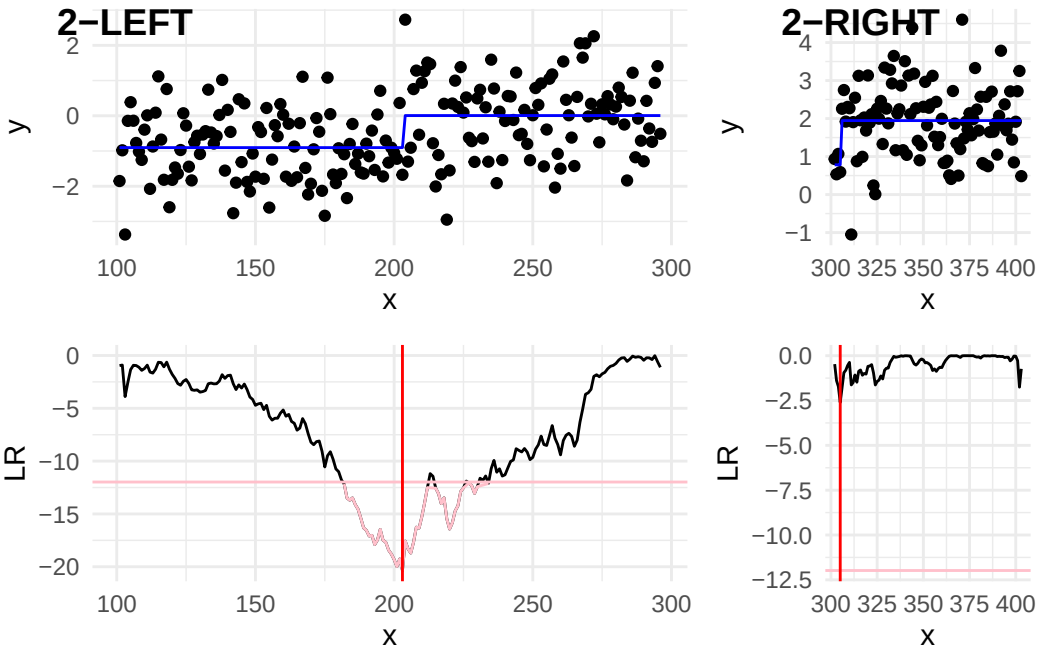
This is not exactly 300, so we don't have a perfect estimate. Despite this is not ideal, this is the best point we have found and therefore we have to continue!



Step 3:

In step 3, we have to check again two splits splits:

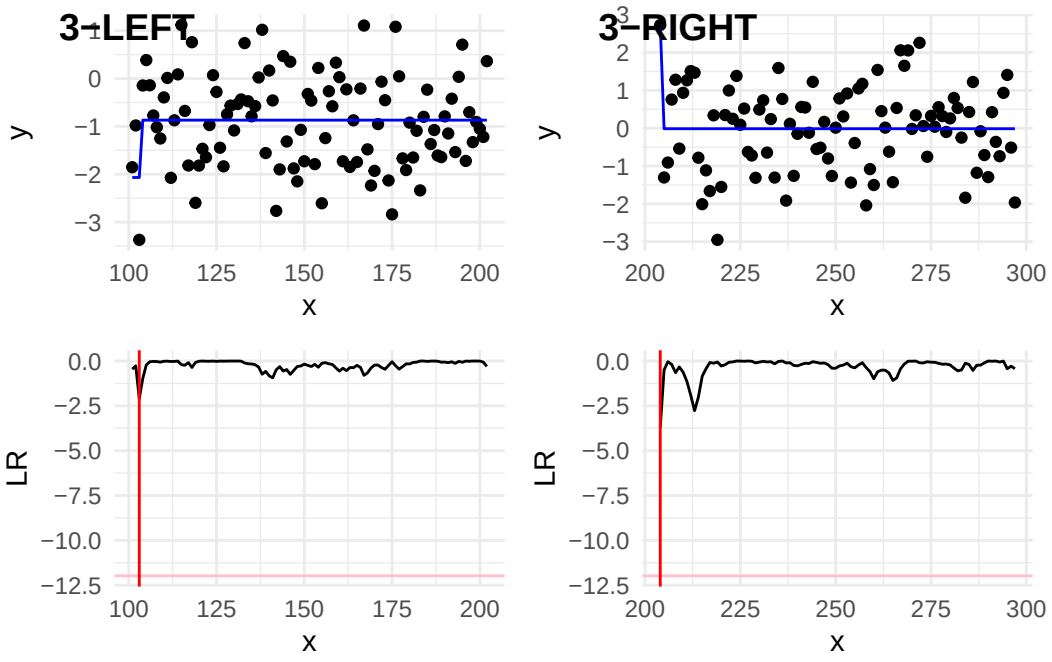
- The second left split, **2-LEFT** in the plot below, covers data $y_{101:297}$. Now, it's in this split that the statistics goes below the threshold! The third estimated change is at $\hat{\tau} = 203$, again slightly off the real one at 200. We continue investigating this split…
- The second right split, **2-RIGHT** covers data $y_{298:400}$. In this last split, the min is not over the threshold, therefore we stop the search.



Step 4:

In step 4, we check:

- The third left split, **3-LEFT** in the plot below, covers data $y_{101:203}$. The minimum, in here is not over the threshold.
- The third right split, **3-RIGHT** covers data $y_{204:298}$. Similarly, the minimum is not over the treshold.



The algorithm therefore terminates!

With this graphical description in mind, we formally describe the Binary Segmentation algorithm as a recursive procedure, where the first iteration would be simply given by $\text{BinSeg}(y_{1:n}, \beta)$.

$\text{BinSeg}(y_{s:t}, \beta)$

INPUT: Subseries $y_{s:t} = \{y_s, \dots, y_t\}$ of length $t - s + 1$, penalty β
OUTPUT: Set of detected changepoints cp

IF $t - s \leq 1$
  RETURN {} // No changepoint in segments of length 1 or less

COMPUTE
$\mathcal{Q} \leftarrow \min_{\tau \in \{s,\dots,t-1\}} [\mathcal{L}(y_{s:\tau}) + \mathcal{L}(y_{\tau+1:t}) - \mathcal{L}(y_{s:t}) + \beta]$

IF $\mathcal{Q} < 0$

$\hat{\tau} \leftarrow \underset{\tau \in \{s,\ldots,t-1\}}{\arg\min} \left[\mathcal{L}(y_{s:\tau}) + \mathcal{L}(y_{\tau+1:t}) - \mathcal{L}(y_{s:t})\right]$
$cp \leftarrow \{\hat{\tau}, \text{BinSeg}(y_{s:\hat{\tau}}, \beta), \text{BinSeg}(y_{\hat{\tau}+1:t}, \beta)\}$
RETURN cp

RETURN {} // No changepoint if $-LR/2$ is above penalty $-\beta$

3.3 Optimal Partitioning

Another solution to avoid the over-fitting problem of Equation 3.1 lies in introducing a penalty term that discourages too many changepoints, avoiding overfitting. This is known as the *penalised approach*.

To achieve this, we want to minimize the following cost function:

$$Q_{n,\beta} = \min_{K \in \mathbb{N}} \left[\min_{\tau_1,\ldots,\tau_K} \sum_{k=0}^{K} \mathcal{L}(y_{\tau_k+1:\tau_{k+1}}) + \beta K \right], \tag{3.3}$$

where $Q_{n,\beta}$ represents the optimal cost for segmenting the data up to time n with a penalty β that increases with each additional changepoint K. With the β term, for every new changepoint added, the cost of the full segmentation increases, discouraging therefore models with too many changepoints.

Unlike Binary Segmentation, which works iteratively and makes local decisions about potential changepoints, and as we have seen it is prone to errors, solving $Q_{n,\beta}$ ensures that the segmentation is **globally optimal**, as in the location of the changes are the best possible to minimise our cost.

Now, directly solving this problem using a brute-force search is computationally prohibitive, as it would require checking every possible combination of changepoints across the sequence: the number of possible segmentations grows exponentially as n increases…

Fortunately, this problem can be solved efficiently using a sequential, dynamic programming algorithm: **Optimal Partitioning (OP)**, from Jackson et al. (2005). OP solves Equation 3.3 exactly through the following recursion.

We start with $\mathcal{Q}_{0,\beta} = -\beta$, and then, for each $t = 1, \ldots, n$, we compute:

$$\mathcal{Q}_{t,\beta} = \min_{0 \le \tau < t} \left[\mathcal{Q}_{\tau,\beta} + \mathcal{L}(y_{\tau+1:t}) + \beta\right]. \tag{3.4}$$

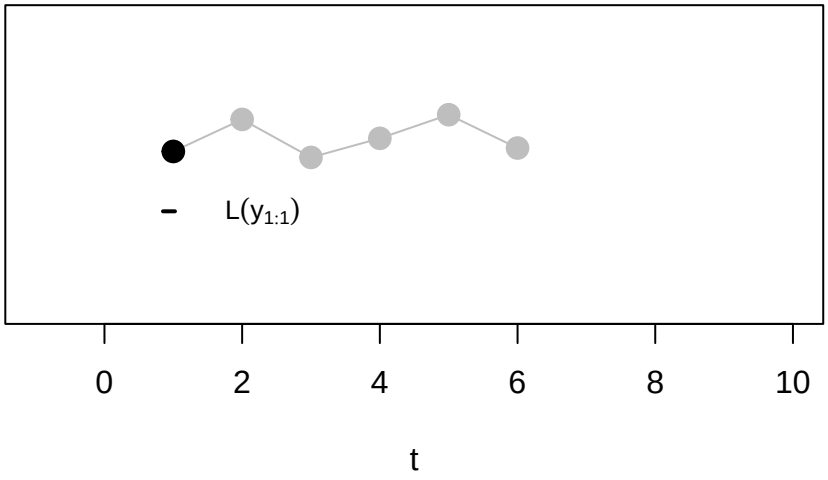
Here, $\mathcal{Q}_{t,\beta}$ represents the optimal cost of segmenting the data up to time t. The algorithm builds this solution sequentially by considering each possible segmentation $\mathcal{Q}_{0,\beta}, \cdots, \mathcal{Q}_{t-2,\beta}, \mathcal{Q}_{t-1,\beta}$ before the current time t, plus the segment cost up to current time t, $\mathcal{L}(y_{\tau+1:t})$.

3.3.1 Optimal partitinioning in action

This recursion can be quite hard to digest, and is, as usual, best described graphically.

Step 1 Say we are at $t = 1$. In this case, according to equation above, the optimal cost up to time one will be given by (remember that the β cancels out with $Q_{0,\beta}$!):

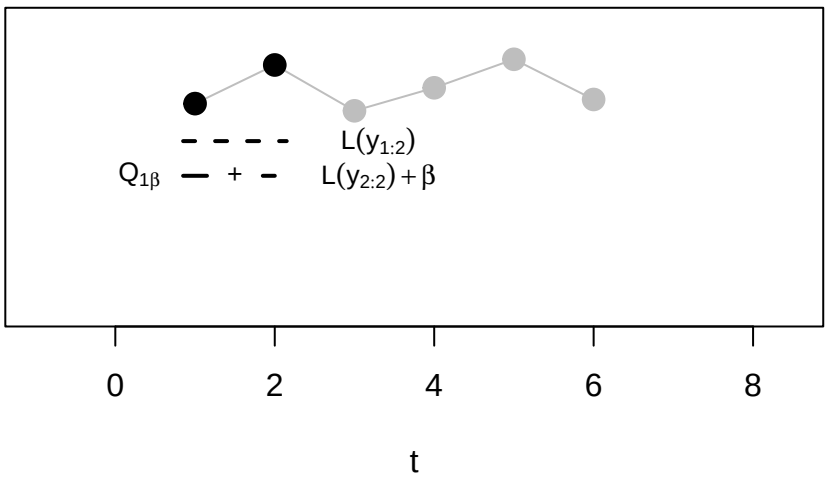
$$\mathcal{Q}_{1,\beta} = [-\beta + \mathcal{L}(y_{1:1}) + \beta] = \mathcal{L}(y_{1:1})$$



Step 2. Now, at the second step, we have to minimise between two segmentations:

- One with the whole sequence in a second segment alone (again, β cancels out with $Q_{0,\beta} = -\beta$), and this will be given by $\mathcal{L}(y_{1:2})$ (dotted line)
- One with the optimal segmentation from step 1 $\mathcal{Q}_{1,\beta}$ (whose cost considered only the first point in its own segment!), to which we have to sum the cost relative to a second segment $\mathcal{L}(y_{2:2})$ that puts the second point alone, and the penalty β as we have added a new segment!

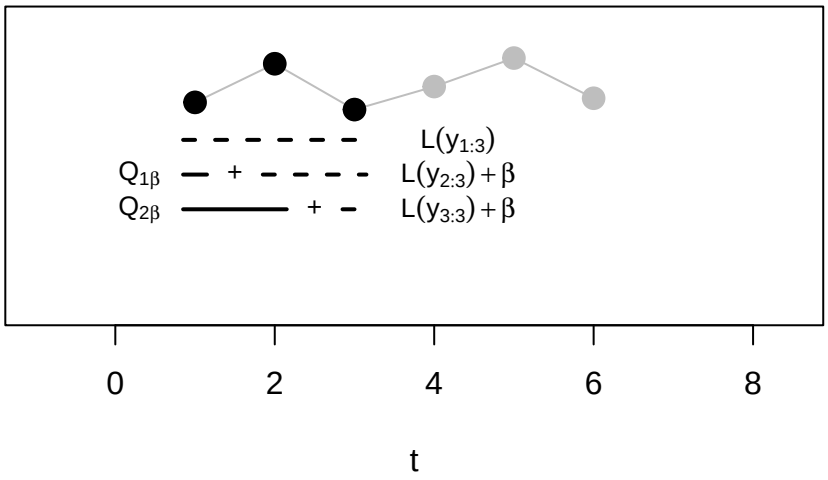
We minimise across the two, and this gives us $Q_{2,\beta}$.



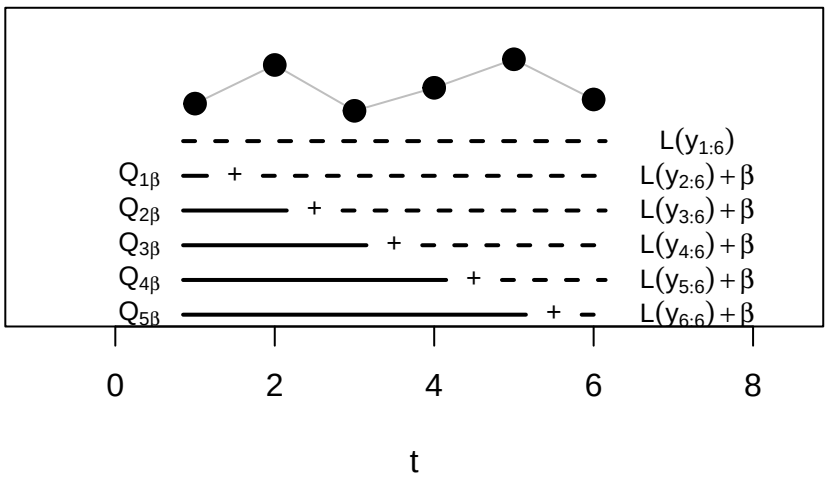
Step 3: Similarly, at $t = 3$ we have now three segmentations to choose from:

- The one that puts the first three observations in the same segment, whose cost will be given simply by $\mathcal{L}(y_{1:2})$,
- The one considering the optimal segmentation from time 1, plus the cost of adding an extra segment with observation 2 and 3 together
- Finally the optimal from segmentation 2, $\mathcal{Q}_{2,\beta}$, plus the segment cost of fitting an extra segment with point 3 alone. Note that $\mathcal{Q}_{2,\beta}$ will come from the step before: if we would have been beneficial to add a change, at the previous step, this information is carried over!

Again, we pick the minimum across these three to get $\mathcal{Q}_{3,\beta}$, and proceed.



Step n Until the last step! Which would look something like this:



A formal description of the algorithm can be found below:

INPUT: Time series $y = (y_1, ..., y_n)$, penalty β
OUTPUT: Optimal changepoint vector cp_n

Initialize $\mathcal{Q}_0 \leftarrow -\beta$
Initialize $cp_0 \leftarrow \{\}$ // a set of vectors ordered by time

FOR $t = 1, \dots, n$
$\mathcal{Q}_t \leftarrow \min_{0 \leq \tau < t} [\mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) + \beta]$
$\hat{\tau} \leftarrow \arg\min_{0 \leq \tau < t} [\mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) + \beta]$
$cp_t \leftarrow (cp_{\hat{\tau}}, \hat{\tau})$ // Append the changepoint to the list at the last optimal point

RETURN cp_n

Note. To implement the line $\mathcal{Q}_t \leftarrow \min_{0 \leq \tau < t} [\mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) + \beta]$, we could either use an inner for cycle and iteratively compute $\mathcal{Q}_t$ for each τ, or we could use a vectorized approach. If we created a vectorized version of our function, it would look something like this:

```
costs = map_dbl(1:t, \(tau) Q[tau] + L(y[(tau+1):t]) + beta)
Q[t + 1] = max(costs)
tau_hat = argmax(costs)
```

We range across `1:t` as R index starts from 1. Remeber that the `map_dbl()` function is used to apply a function to each element of a vector and return a new vector. This would be more efficient and faster. We will code the Optimal Partitioning algorithm in the lab.

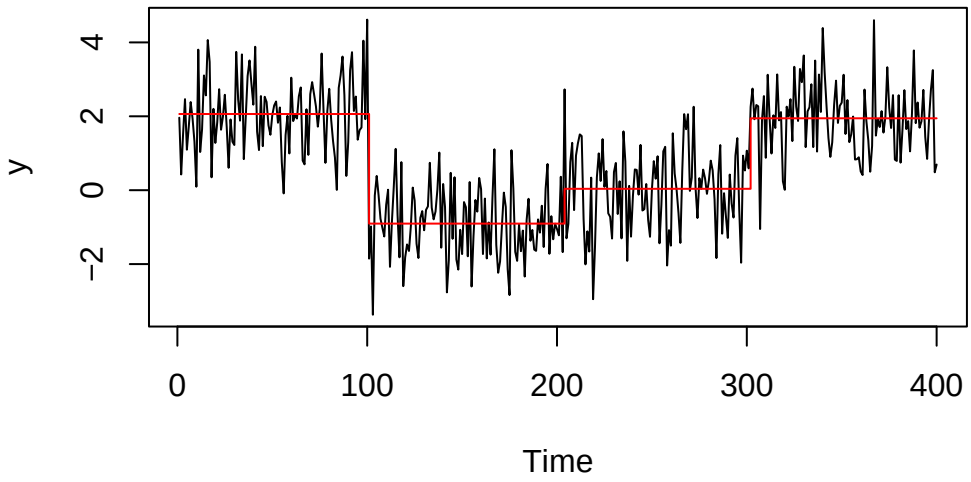
Running the Optimal Partitioning method on our example scenario, with the same penalty $\beta = 2\log(400) = 11.98$ as above, gives changepoint locations $\tau_{1:4} = \{100, 203, 301\}$.

```
Loading required package: zoo
```

```
Attaching package: 'zoo'
```

```
The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

```
Successfully loaded changepoint package version 2.3
 WARNING: From v.2.3 the default method in cpt.* functions has changed from AMOC to PELT.
 See NEWS for details of all changes.
```

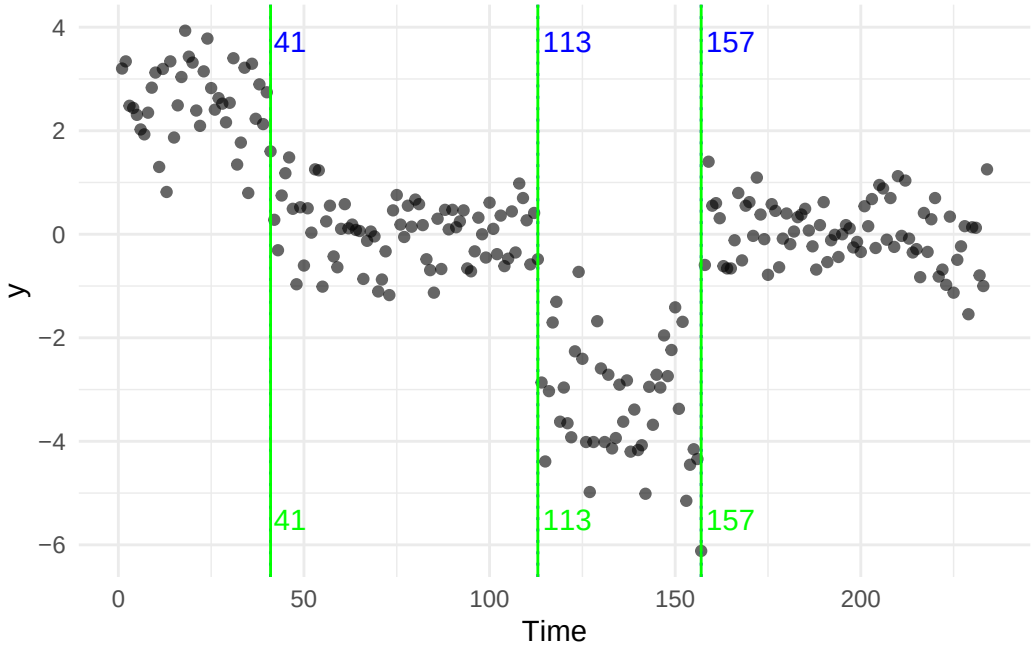


So we can see how on this dataset in particular, OP performs slightly better then Binary Segmentation on the last change, getting closer to the real changepoint of 300!

3.3.2 Neuroblastoma example

Returning to the original example at the start of the module, the neuroblastoma dataset, we run both Binary Segmentation, and Optimal Partitioning.

We report results in the plot below (blue for BS, green for OP). In this case, the algorithms return the same four changepoints:



Some of you might come up with two (very interesting) questions that hopefully we will answer next week...

- If the methods perform roughly the same, which one do I choose?
- Why is the data on a different scale then that presented at the start of the chapter?

3.4 Exercises

3.4.1 Workshop 3

1. For the vector $y_{1:4} = (0.5, -0.1, 12.1, 12.4)$, and a penalty $\beta = 5$ calculate, pen on paper (and calculator), all the Optimal Partitioning and Binary Segmentation steps. **TIP**: To speed up computations, you want to pre-compute all segment costs $\mathcal{L}(y_{l:u})$. I have pre-computed some of these costs in the table below:

$l \backslash u$	1	2	3	4
1	$\mathcal{L}(y_{1:1})$	0.18	94.59	145.43
2		0.00	$\mathcal{L}(y_{2:3})$	101.73
3			0.00	$\mathcal{L}(y_{3:4})$
4				0.00

3.4.2 Lab 3

1. Code the Optimal Partitioning algorithm for the Gaussian change-in-mean case.

Your function should take as input three things:

- A vector `y`, our observations
- A double `penalty`, corresponding to the β penalty of our penalised cost
- A function `COST`. This function should take as input arguments `y, s, t`, and act as $\mathcal{L}(y_{s:t})$. You fill find a skeleton below.

```
OP <- function (y, penalty, COST) {

  ### pre-compute all the costs here

  ### your initialization here

  for (t in 1:n) {

    ### your recursion here

  }


  return(changepoints)
}
```

For this exercise, we are implementing the Gaussian change-in-mean case. Therefore, a skeleton for our cost function will be:

```
costMean = function(y, s, t) {

  # code for computing the Gaussian change-in-mean cost here
```

```
  return(your_cost)
}
```

You should be then able to call your `OP` function as:

```
set.seed(123)
y <- c(rnorm(100), rnorm(100, 5), rnorm(100, -1))
OP(y, 15, costMean)
```

This should return 100, 200.

```
**Tips:**

a.  Again, you can pre-compute all the possible $\mathcal{L}(y_{1:u})$, for $u \geq 1$ to sa

b.  Be very careful with indexing... R starts indexing at 1, however, in the pseudocode, you
```

4 PELT, WBS and Penalty choices

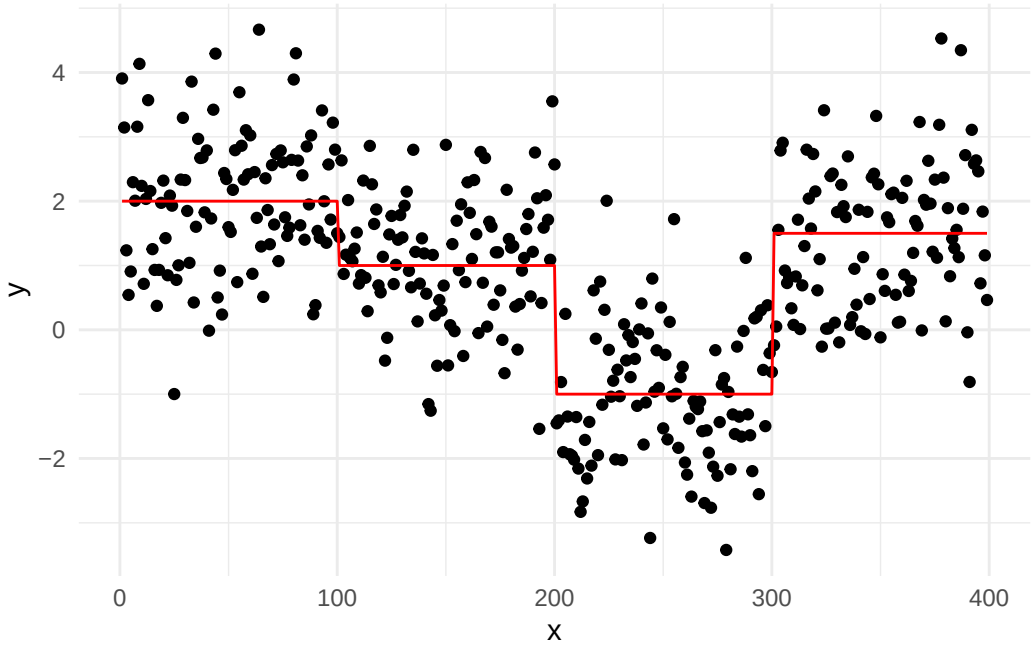
4.1 Drawbacks of OP and BS

When deciding which segmentation approach to use, Binary Segmentation (BS) and Optimal Partitioning (OP) each offer different strengths. The choice largely depends on the characteristics of the data and the goal of the analysis.

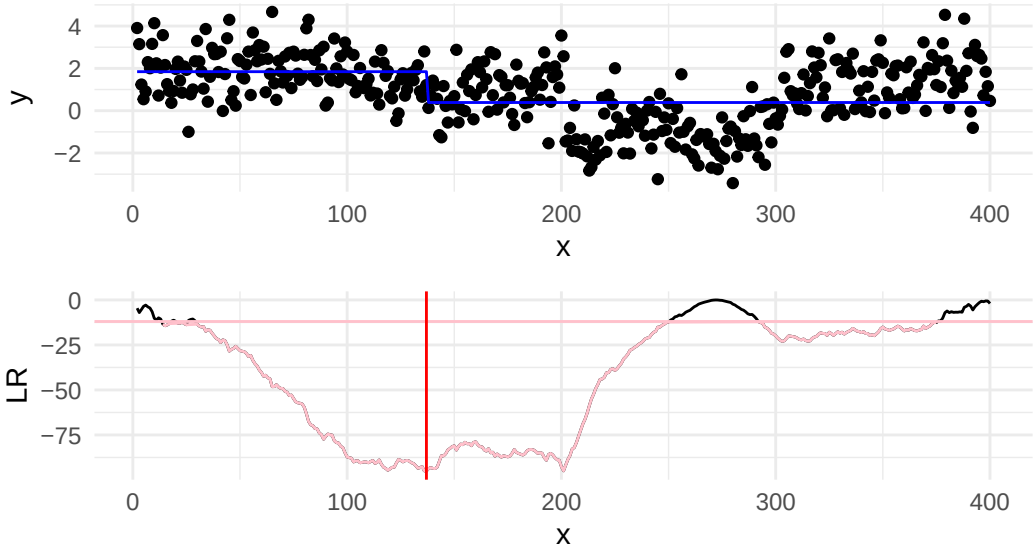
4.1.1 Quality of the Segmentation

Generally, Optimal Partitioning (OP) provides the *most accurate segmentation*, especially when we have a well-defined model and expect precise changepoint detection. OP ensures that the solution is optimal by globally minimizing the cost function across all possible segmentations. This is ideal for datasets with clear changes, even if noise is present.

Let's consider a case with true changepoints at $\tau = 100, 200, 300$, and segment means $\mu_{1:4} = 2, 1, -1, 1.5$:

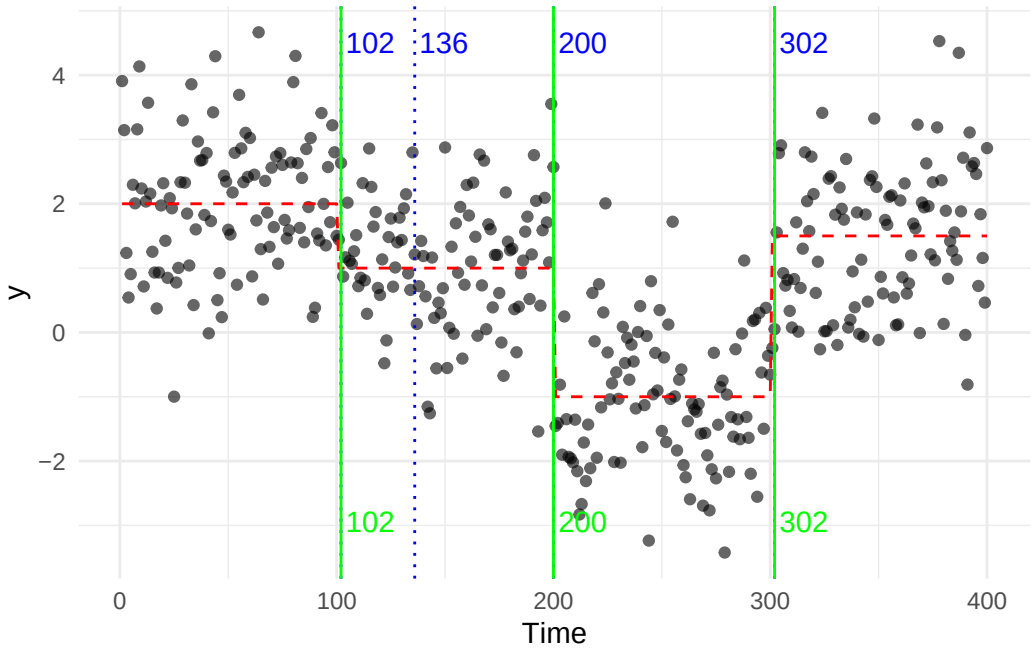


While the underlying signal follows these clear shifts, noise complicates segmentation. Binary Segmentation uses a greedy process where each iteration looks for the largest changepoint. Although fast, this local search can make mistakes if the signal isn't perfectly clear, particularly in the early stages of the algorithm. For example, running BS on this dataset introduces a mistake at $\tau = 136$, as shown in the plot below:



This error is carried in the subsequent steps, and the full binary segmentation algorithm will output an additional change at $\tau = 136$... Optimal Partitioning (OP), on the other hand, evaluates all possible segmentations considers the overall fit across the entire sequence. It is therefore less susceptible to adding "ghost" changepoints, as rather than focusing on the largest change at each step.

To illustrate, we compare the segmentations generated by both approaches:



4.1.2 Computational Complexity

Well, you may ask why not using OP all the time, then? Well, in changepoint detection, in which is the most appropiate method, we often have to keep track of the computational performance too, and Binary Segmentation is faster on average. For this reason, for large datasets where approximate solutions are acceptable, it might be the best option.

Specifically:

- **Binary Segmentation** starts by dividing the entire sequence into two parts, iteratively applying changepoint detection to each segment. In the average case, it runs in $\mathcal{O}(n \log n)$ because it avoids searching every possible split point. However, in the worst case (if all data points are changepoints), the complexity can degrade to $\mathcal{O}(n^2)$, as each step can require recalculating test statistics for a growing number of segments.
- **Optimal Partitioning**, on the other hand, solves the changepoint problem by recursively considering every possible split point up to time t. The result is an optimal segmentation, but at the cost of $\mathcal{O}(n^2)$ computations. This holds true for both the average and worst cases, as it always requires a full exploration of all potential changepoints.

4.2 PELT and WBS

Good news is, despite both algorithms have drawbacks, following *recent developments*, those have been solved. In the next sections, we will introduce two new algorithms, PELT and WBS.

4.2.1 PELT: an efficient solution to OP

In OP, we can reduce the numbers of checks to be performed at each iteration, reducing the complexity. This operation is called *pruning*. Specifically, given a cost function $\mathcal{L}(\cdot)$, on the condition that there exists a constant κ such that for every $l < \tau < u$:

$$\mathcal{L}(y_{l+1:\tau}) + \mathcal{L}(y_{\tau+1:u}) + \kappa \leq \mathcal{L}(y_{l+1:u})$$

It is possible to *prune* without resorting to an approximation. For many cost functions, such as the Gaussian cost, such constant κ exists.

Then, for any $\tau < s$, if

$$\mathcal{Q}_{\tau,\beta} + \mathcal{L}(y_{\tau+1:t}) + \kappa \geq \mathcal{Q}_{t,\beta}$$

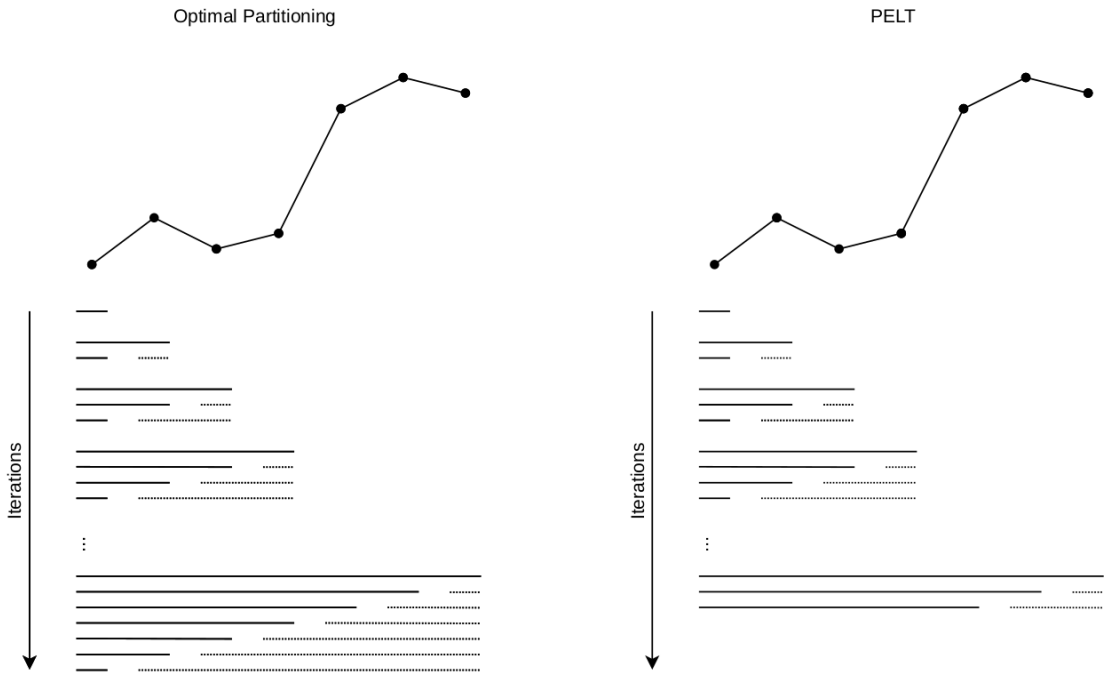
holds, then for any $T > t$, τ can never be the optimal change location prior to time T.

Using this condition, the PELT algorithm – acronym for Pruned Exact Linear Time – (Killick, Fearnhead, and Eckley (2012)) solves exactly the penalised minimization of Equation 3.4 with an expected computational cost that can be linear in n – while still retaining $\mathcal{O}(n^2)$ computational complexity in the worst case. This is achieved by reducing the number of segment costs to evaluate at each iteration via an additional pruning step based on Condition Equation 3.4. That is, by setting $\kappa = 0$, if

$$\mathcal{Q}\tau + \mathcal{L}(y_{\tau+1:t}) \geq \mathcal{Q}_t$$

then we can safely prune the segment cost related to τ, as τ will never be the optimal change-point location up to any time $T > t$ in the future.

The intuition, is that we would prune at every change detected. And if the changes increase linearly with the length of the data, this means that our algorithm will achieve a $\mathcal{O}(n \log n)$ computational complexity, without any drawbacks!



To reduce computational complexity, we can slightly modify the OP algorithm, to add the pruning condition above:

PELT

INPUT: Time series $y = (y_1, ..., y_n)$, penalty β
OUTPUT: Optimal changepoint vector cp_n

Initialize $\mathcal{Q}_0 \leftarrow -\beta$
Initialize $cp_0 \leftarrow \{\}$ // a set of vectors ordered by time
Initialise $R_1 = (0)$ // a vector of candidate change locations

FOR $t = 1, \dots, n$
$\mathcal{Q}_t \leftarrow \min_{\tau \in R_t} [\mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) + \beta]$
$\hat{\tau} \leftarrow \arg\min_{\tau \in R_t} [\mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) + \beta]$
$cp_t \leftarrow (cp_{\hat{\tau}}, \hat{\tau})$ // Append the changepoint to the list at the last optimal point
$R_{t+1} \leftarrow \{\tau \in R_t : \mathcal{Q}_\tau + \mathcal{L}(y_{\tau+1:t}) < \mathcal{Q}_t\}$ // select only the change locations that are still optimal
$R_{t+1} \leftarrow (R_{t+1}, t)$ // add t to the points to check at the next iteration

RETURN cp_n

As the segmentation retained is effectively the same, there are literally no disadvantages in using PELT over OP, if the cost function allows to do so.

However, PELT still has some disadvantages:

- PELT pruning works only over some cost functions, those for which the condition above is true. For example, in a special case of change-in-slope, as we will see in the workshop, we have that the cost from the next change depends on the location of the previous one, making it impossible for PELT to prune without loosing optimality.
- We mentioned above how PELT over iterations at which a change is detected. For signals where changes are not frequent, PELT does not benefits from. A more sophisticated approach is that of **FPOP**, that prunes at every iteration. FPOP employs a different type of pruning, called functional pruning, that at every iteration only check costs that are likely associated to a change. However, despite the pruning is stronger FPOP works only over few selected models.

4.2.2 WBS: Improving on Binary Segmentation

In BS, one of the issues that may arise, is an incorrect segmentation. WBS, Fryzlewicz (2014), is a multiple changepoints procedures that improve on the BS changepoint estimation via computing the initial segmentation cost of BS multiple times over $M + 1$ random subsets of the sequence, $y_{s_1:t_1}, \dots, y_{s_M:t_M}, y_{1:n}$, picking the best subset according to what achieves the smallest segmentation cost and reiterating the procedure over that sample accordingly. The idea behind WBS lies in the fact that a favorable subset of the data $y_{s_m:t_m}$ could be drawn which contains a true change sufficiently separated from both sides s_m, t_m of the sequence. By the inclusion of the $y_{1:n}$ entire sequence among the subsets, it is guaranteed that WBS will do no worse than the simple BS algorithm.

We can formally provide a description of WBS as a recursive procedure.

We first start by drawing the set of intervals $\mathcal{F} = \{[s_1, t_1], \dots, [s_M, t_M]\}$ where:

$$s_m \sim \mathrm{U}(1, n),\ t_m \sim \mathrm{U}(s_m + 1, n)$$

Then, WBS will have just a couple of alterations to the original Binary Segmentation:

WBS($y_{s:t}, \beta$)

INPUT: Subseries $y_{s:t} = \{y_s, \dots, y_t\}$ of length $t - s + 1$, penalty β
OUTPUT: Set of detected changepoints cp

IF $t - s \leq 1$
  RETURN {} // No changepoint in segments of length 1 or less

$\mathcal{M}_{s,e}$ = Set of those indices m for which $[s_m, t_m] \in \mathcal{F}$ is such that $[s_m, t_m] \subset [s, t]$.
$\mathcal{M} \leftarrow \mathcal{M} \cup \{[1, n]\}$
COMPUTE
$$\mathcal{Q} \leftarrow \min_{\substack{[s_m,t_m]\in\mathcal{M} \\ \tau\in\{s_m,\dots,t_m\}}} [\mathcal{L}(y_{s:\tau}) + \mathcal{L}(y_{\tau+1:t}) - \mathcal{L}(y_{s:t}) + \beta]$$

IF $\mathcal{Q} < 0$
  $$\hat{\tau} \leftarrow \arg\min_{\substack{[s_m,t_m]\in\mathcal{M} \\ \tau\in\{s_m,\dots,t_m\}}} [\mathcal{L}(y_{s:\tau}) + \mathcal{L}(y_{\tau+1:t}) - \mathcal{L}(y_{s:t})]$$
  $cp \leftarrow \{\hat{\tau}, \text{WBS}(y_{s:\hat{\tau}}, \beta), \text{WBS}(y_{\hat{\tau}+1:t}, \beta) + \hat{\tau}\}$
  RETURN cp

RETURN {} // No changepoint if $-LR/2$ is above penalty $-\beta$

One of the major drawbacks of WBS is that in scenarios where we find frequent changepoints, in order to retain a close-to-optimal estimation, one should draw a higher number of M intervals (usually of the order of thousands of intervals). This can be problematic given that WBS has computational complexity that grows linearly in the total length of the observations of the subsets.

4.3 Penalty Selection

In previous sections, we applied the changepoint detection algorithms using a penalty term of $2\log(n)$. As we'll see, this is the **BIC penalty** (Bayes Information Criterion), a widely used penalty in changepoint detection. However, it is important to note that BIC is just one of several penalty types that can be applied…

As in the single change, some penalty may be more conservative then others! Choosing the correct penalty is key to obtaining a sensible segmentation of the data. The penalty term plays a significant role in balancing the goodness-of-fit of the model with its complexity:

- A lower penalty may lead to an over-segmentation, where too many changepoints are detected
- A higher penalty could under-segment the data, missing important changepoints.

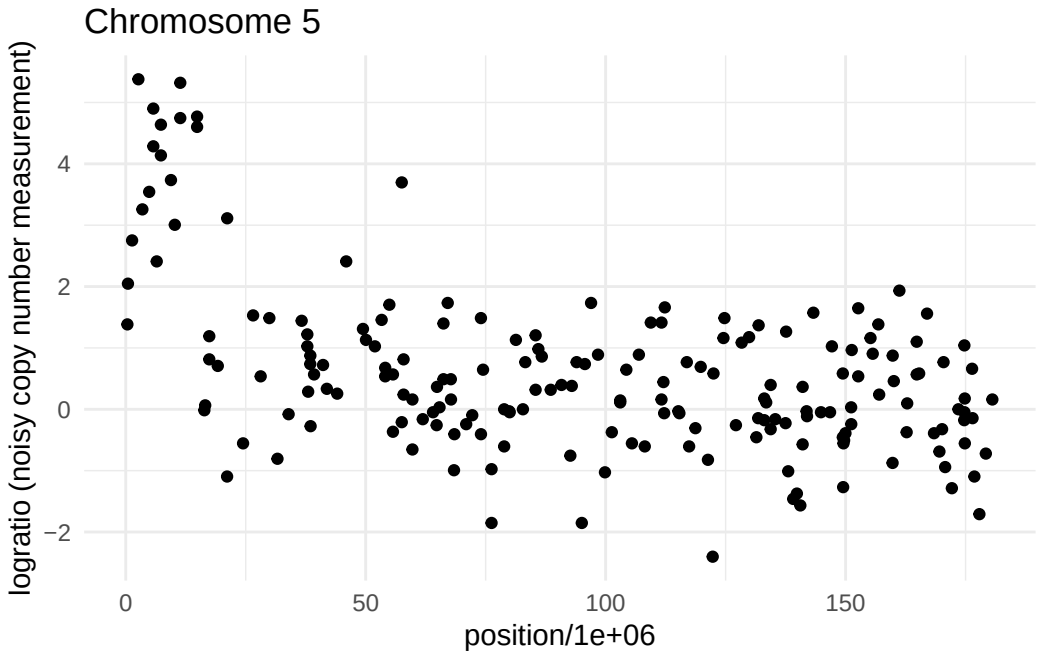
The three most common penalties, are:

- **AIC (Akaike Information Criterion):** The AIC penalty takes value of $2p$, where p is the number of parameters that one adds to the model. In multiple changes scenario, every new change, we add a new parameter to the model (as we estimate the signal). This, in OP and BS approaches, where the penalty is added at different iterations, shouls we fit a change, this translates in $\beta = 2 \times 2 = 4$ as our β. While simple to apply, AIC is known to be *asymptotically inconsistent*: it tends to overestimate the number of changepoints as the sample size increases. Intuitively, this is because AIC is designed to minimize the prediction error rather than to identify the true model structure. It favors models that fit the data well, often leading to the inclusion of more changepoints than necessary.
- **BIC (Bayesian Information Criterion):** The BIC penalty is given by $p \log(n)$. In our approaches, this translates to: $\beta = 2 \log(n)$, that we add for each additional changepoint. BIC is generally more conservative than AIC and is consistent, meaning it will not overestimate the number of changepoints as the sample size grows.
- **MBIC (Modified BIC):** The MBIC penalty, from Zhang and Siegmund (2007), is an extension of the BIC that includes an extra term to account for the spacing of the changepoints. We can approximate it, in practice, by using a value of $\beta = 3 \log(n)$ as our penalty. In practice, it is even more conservative then the BIC penalty.

4.3.1 Example in R: Comparing Penalties with PELT

Let's now examine how different penalties impact the results of changepoint detection using the `changepoint` package in R. We'll focus on the PELT method and compare the outcomes when using AIC, BIC, and MBIC penalties.

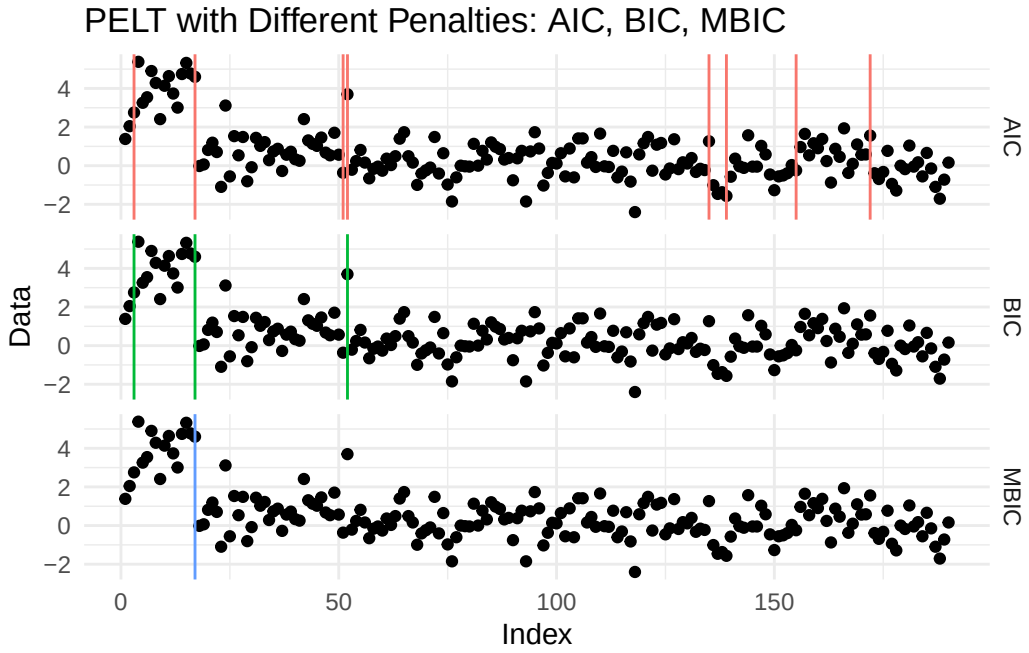
As a data sequence, we will pick a different chromosome in our Neuroblastoma dataset. Can you tell, by eye, how many changes are in this sequence?



We can compare the three penalties using the changepoint library, as below:

```
data <- one.dt$logratio
n <- length(data)

# Apply PELT with AIC, BIC, and MBIC penalties
cp_aic <- cpt.mean(data, method = "PELT", penalty = "AIC")
cp_bic <- cpt.mean(data, method = "PELT", penalty = "BIC")
cp_mbic <- cpt.mean(data, method = "PELT", penalty = "MBIC")

# Extract changepoint locations for each penalty
cp_aic_points <- cpts(cp_aic)
cp_bic_points <- cpts(cp_bic)
cp_mbic_points <- cpts(cp_mbic)

# Create a data frame for plotting with ggplot2
plot_data <- data.frame(
  index = 1:n,
  data = data)

# Create data frames for changepoints with corresponding method labels
cp_df <- bind_rows(
  data.frame(index = cp_aic_points, method = "AIC"),
  data.frame(index = cp_bic_points, method = "BIC"),
```

```
  data.frame(index = cp_mbic_points, method = "MBIC")
)

ggplot(plot_data, aes(x = index, y = data)) +
  geom_point() +  # Plot the data line first
  geom_vline(data = cp_df, aes(xintercept = index, color = method)) +
  facet_grid(method ~ .) +
  labs(title = "PELT with Different Penalties: AIC, BIC, MBIC", x = "Index", y = "Data") +
  theme_minimal() +
  theme(legend.position = "none")
```



We can see how from this example, the AIC likely overestimated the number of changepoints, while BIC and MBIC provided more conservative and reasonable segmentations. By eye, the MBIC seems to have done the better job!

4.3.2 CROPS: running with multiple penalties

Hopefully, the example above should have highlighted that finding the right penalty can be tricky. One solution, would be to run our algorithm for a range of penalties, and then choose a posteriori what the best segmentation is. The CROPS algorithm, from Haynes, Eckley, and Fearnhead (2017), is based on this idea. CROPS works alongside an existing penalised changepoint detection algorithm, like PELT or WBS: as long as the changepoint method can map a penalty value to a (decreasing) segmentation cost, CROPS could be applied.

CROPS takes as input a range of penalties $[\beta_{\min}, \beta_{\max}]$, and explores all possible segmentations within those two penalties in a clever way, to fit the changepoint model as least as we can. As CROPS calculates changepoints for a particular penalty, it keeps track of the range of penalty values where that specific set of changepoints is valid. This works because, for certain ranges of penalties, the set of changepoints stays the same.

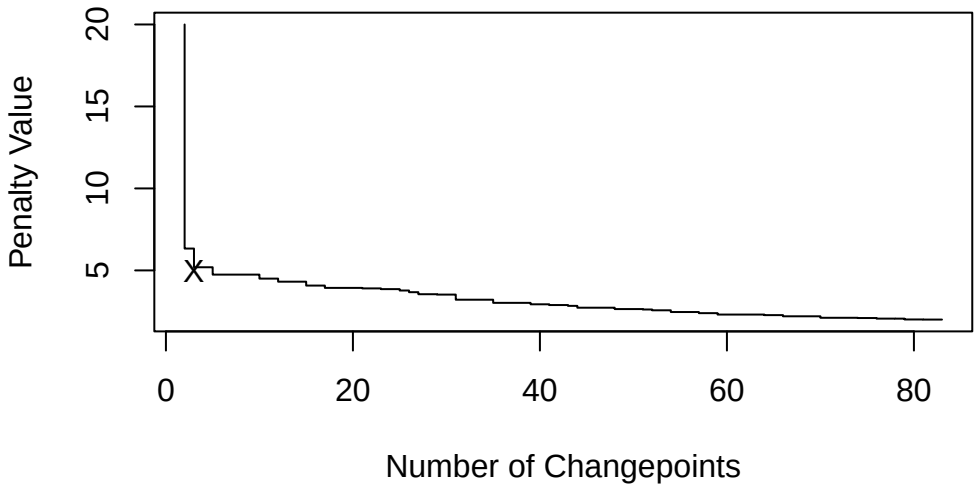
E.g. for penalties between β_1 and β_2, the changepoints might remain the same, so CROPS only needs to run the changepoint detection once for that range.

We won't introduce the method formally, but in an intuitive way, CROPS works in this way:

1. It starts calculates changepoints at two extreme penalties: $\beta_{\min}$ and $\beta_{\max}$. If those are the same, it quits.
2. Alternatively, as a binary search, CROPS selects a mid-point penalty β_{int} based on whether the segmentation change, and runs the changepoint detection again on $[\beta_{\min}, \beta_{\text{int}}]$, and $[\beta_{\text{int}}, \beta_{\max}]$, refining its search for the next penalty.
3. It repeats 2 iteratively until no further segmentations are found.

We can use CROPS to generate an **elbow plot** for selecting the appropriate penalty value in changepoint detection. In Data Science and Machine Learning, elbow plots are graphs that helps us choosing the appropiate value of a parameter, balancing between model complexity (in our case number of changepoints) and goodness of fit (how tightly our model fits the data).

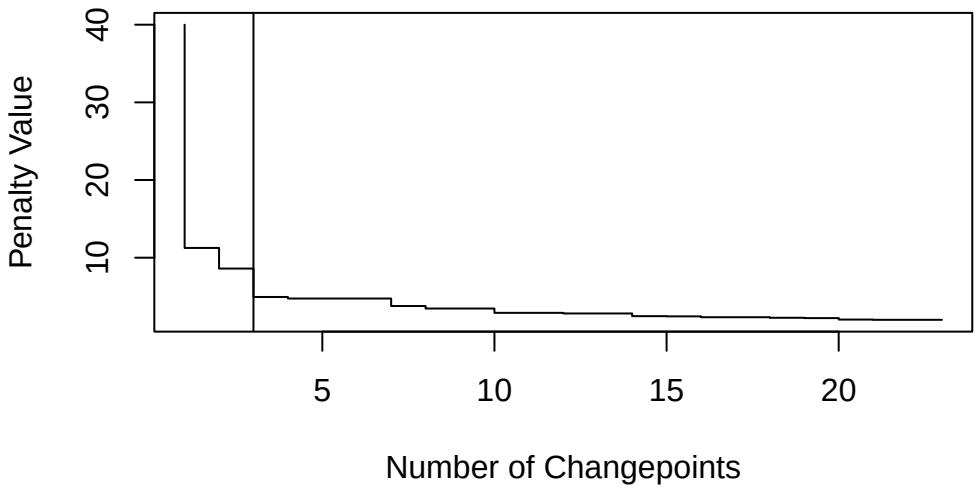
In case of CROPS, we can plot the number of changepoints against the penalty value from our range. The curve typically shows a steep drop at first, as many changepoints are detected with low penalties, then flattens as the penalty increases and fewer changepoints are added. The **elbow** (hence its name) is the point where the rate of change in the number of changepoints significantly slows down:



The elbow is a point of balance between model fit and complexity. As a rule of thumb, a good choices of a penalty reside in picking either the penalty that generates the segmentation at the elbow, or the one at the point immediately prior.

Going back to our neuroblastoma example above. We run CROPS for penalties $[2, 40]$, and we then generate the elbow plot:

```
out <- cpt.mean(data, method = "PELT", penalty  = "CROPS", pen.value = c(2, 40))

plot(out,diagnostic=TRUE)
abline(v=3)
```



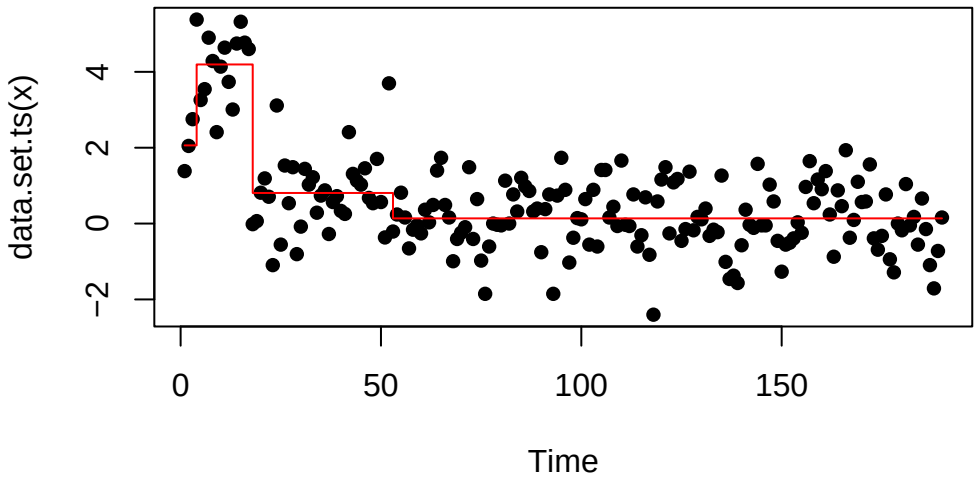
We can see that the elbow is at 4 changepoints, therefore this could suggest that a segmentation with 4 changes might be the best!

This gives us:

```
cpts(out, 3)
```

```
[1]  3 17 52
```

```
plot(out, ncpts= 3, type="p", pch=16)
```



4.4 Exercises

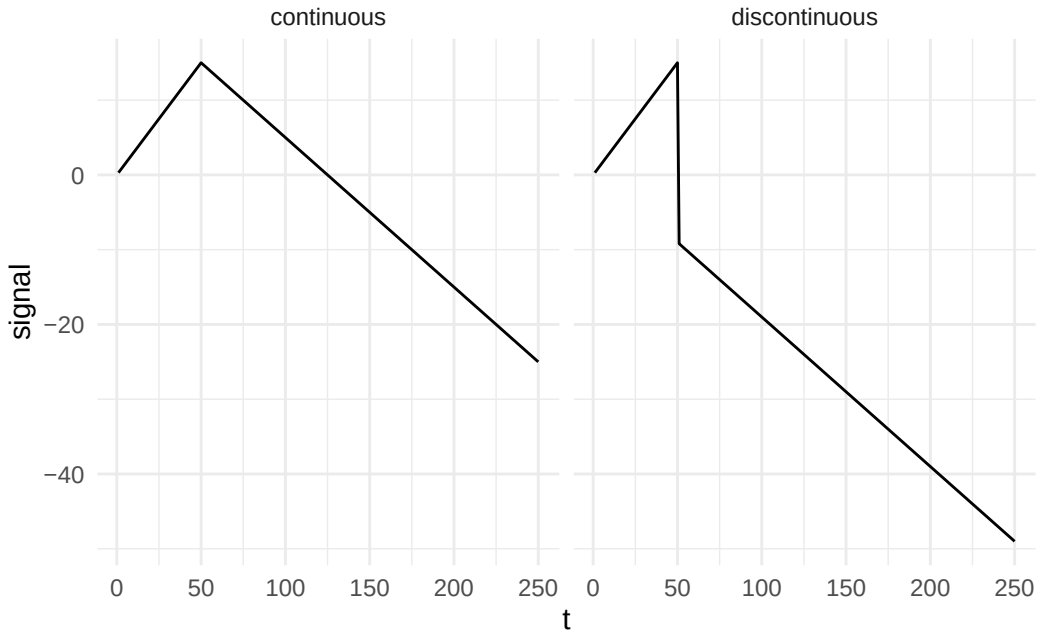
4.4.1 Workshop 4

1. Looking at last week workshop exercise solution, which points in the OP recursion would have been pruned by PELT? Check that the PELT pruning condition is true.
2. The model (not the cost!) for a single segment of a *continuous* change-in-slope is given by:

$$y_t = \tau_i \theta_{\tau_i} + \theta_{\tau_{i+1}}(t - \tau_i) + \epsilon_t, \text{ for } t = \tau_i + 1, \ldots, \tau_{i+1}, \epsilon_t \sim N(0, 1) \tag{4.1}$$

where θ_{τ_i} represents the value of the slope at changepoint τ_i and $\phi_{\tau_{i+1}}$ is the value at the next changepoint τ_{i+1}. Note, in this example, for simplicity, we assume the intercept is set equal to 0.

This model is a variation from the one we had next week as it enforces continuity, e.g. the value at the end of one segment, needs to be the at the next:



a. Can you identify the elements where there is dependency across segments? Once you've done that, rewrite the model for one change by seting $\tau_i = 0$. Would you be able to use PELT pruning with this one?

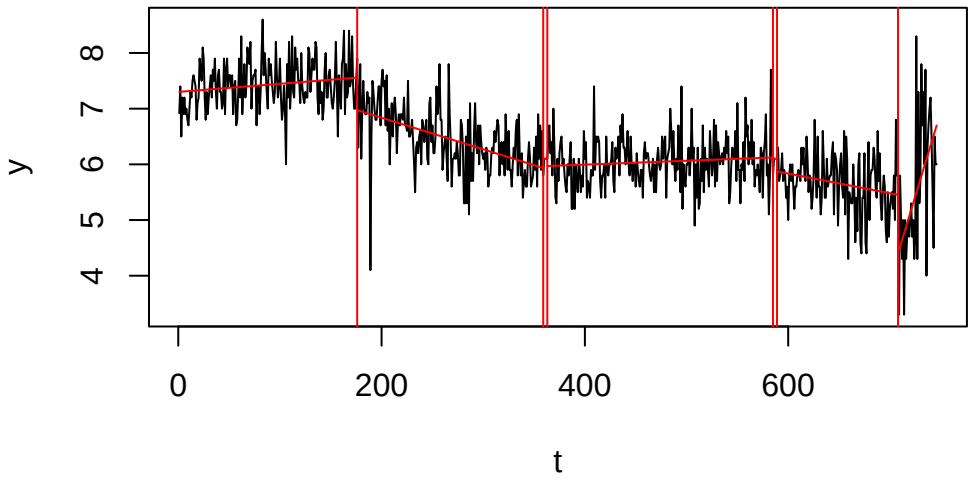
b. Write down the continuous model from equation Equation 4.1, and the one from the previous point for a case where you have two segments, θ_1, θ_2. Then have a look at the model from week 2! What are the differences across the three models?

3. The PELT algorithm will only be able to deal with the discontinuous model. We will now revisit the Simpsons dataset, fitting this multiple changes model. This can be achieved via:

```
library(changepoint)

data <- cbind(y, 1, 1:length(y))
out <- cpt.reg(data, method="PELT")

cat("Our changepoint estimates:", cpts(out))
```

```
Our changepoint estimates: 176 359 363 585 589 708
```

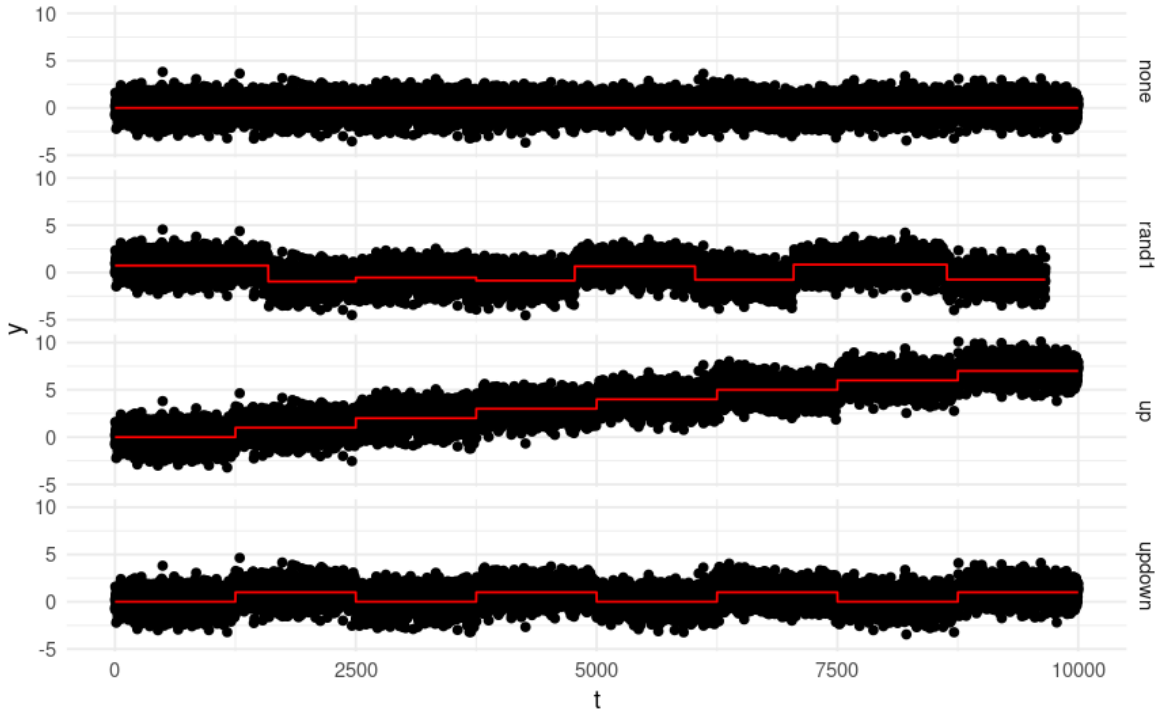
```
plot(out, ylab="y", xlab="t", pch=16)
abline(v = cpts(out), col = "red")
```



Comment this segmentation. In which way we improved from the segmentation in week 2? What would you change?

4.4.2 Lab 4

In this lab we will test changepoint algorithms over some artificial data. Each sequence will have one of the following Gaussian change-in-mean patterns:



The code below will generate 400 sequences, which will be stored in a list called `full_seqs`. Every 100 sequences you will have a different change-pattern, across the four different change patterns.

```
library(tidyverse)

generate_signal <- function(n, pattern = c("none", "up", "updown", "rand1"), nbSeg = 8, jumpS
  type <- match.arg(pattern)

  if (type == "rand1") {
    set.seed(42)
    rand1CP <- rpois(nbSeg, lambda = 10)
    r1 <- pmax(round(rand1CP * n / sum(rand1CP)), 1)
    s <- sum(r1)

    # Adjust r1 to match sum to n
    r1 <- if (s > n) {
      while (sum(r1) > n) r1[which(r1 > 1)[sample(1:length(which(r1 > 1)), 1)]] <- r1[which(r
      r1
    } else {
      sample(rep(seq_along(r1), n - s)) %>% table() %>% as.numeric() %>% `+`(r1)
    }

    set.seed(43)
    rand1Jump <- runif(nbSeg, min = 0.5, max = 1) * sample(c(-1, 1), nbSeg, replace = TRUE)
  }

  # Generate scenarios
  switch(
    type,
    none = rep(0, n),
    up = rep(seq(0, nbSeg - 1) * jumpSize, each = n / nbSeg),
    updown = rep((seq(0, nbSeg - 1) %% 2) * jumpSize, each = n / nbSeg),
    rand1 = map2(rand1Jump, r1, ~rep(.x * jumpSize, .y)) %>% unlist()
  )
}

sims <- expand_grid(pattern = c("none", "up", "updown", "rand1"), rep = 1:100)

full_seqs <- pmap(sims, \(pattern, rep) {
  mu <- generate_signal(1e4, pattern)
  set.seed(rep)
  y <- mu + rnorm(length(mu))
```

```
  cps <- which(diff(mu) != 0)
return(list(y = y, mu = mu, cps = cps, pattern = pattern))
})

# each component of the list describes a sequence:
summary(full_seqs[[1]])
```

```
        Length Class  Mode
y       10000  -none- numeric
mu      10000  -none- numeric
cps         0  -none- numeric
pattern     1  -none- character
```

1. Plot four sample sequences, each with a different change pattern, with superimposed signals. You should replicate the plot above.
2. Install the `changepoint` package. By researching `?cpt.mean`, learn about the change in mean function. Run the PELT algorithm for change in mean on the four sequences you picked above, with MBIC penalty.
3. Compare, in a simulation study, across the four different scenarios, performances of:
    a. Binary Segmentation, with AIC and BIC penalty
    b. PELT, with AIC and BIC penalty

You will need to compare performances in term of Mean Square Error of the fitted signal $\text{MSE} = ||\mu_{1:n} - \hat{\mu}_{1:n}||_2^2$. A function has been already coded for you below:

```
mse_loss <- function(mu_true, mu_hat) {
  return(sum((mu_true - mu_hat) ^ 2))
}
```

Report results by scenario and algorithm.

NOTE: You will be able to access parameters estimates via the function `param.est()`. To get $\hat{\mu}_{1:n}$, necessary for the MSE computation above, we can use:

```
results <- # cpt.mean output here
rep(param.est(result)$mean, times = diff(c(0, cp_est, length(y))))
```

5 Working with Real Data

In practice, working with real-world data presents various challenges that can complicate our analyses. Unlike idealised examples, real data often contain noise, outliers, and other irregularities that can impair the accuracy of the segmentations we aim to generate. The assumptions we make in our models may not hold up well, and this can lead to poor estimates of changepoints. To tackle these issues, it is important to either use robust methods, or consider carefully how we handle the estimation of key parameters within our changepoint detection models.

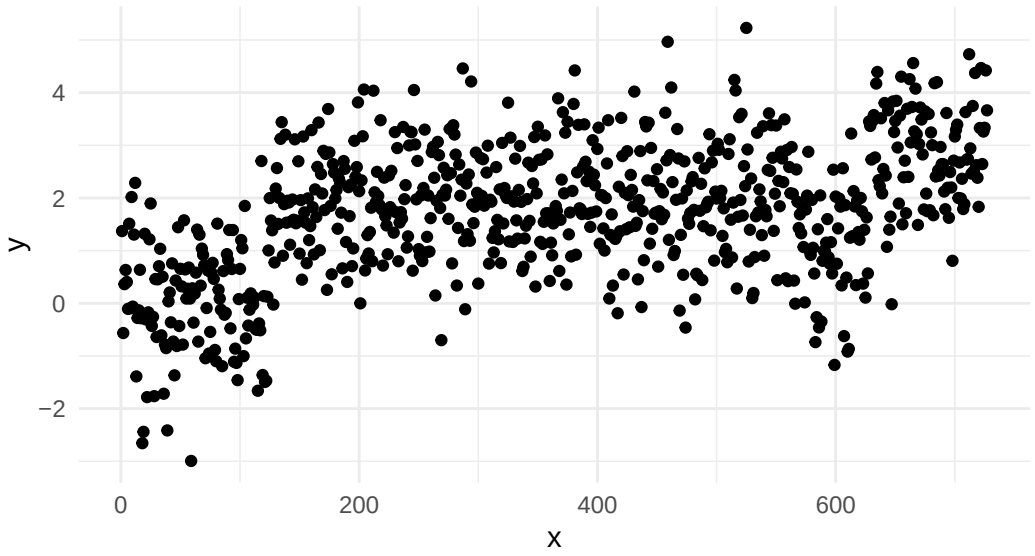
5.1 Assessing the model fit

5.1.1 Assessing Residuals from a Changepoint Model: A Guide for Undergraduate Mathematics

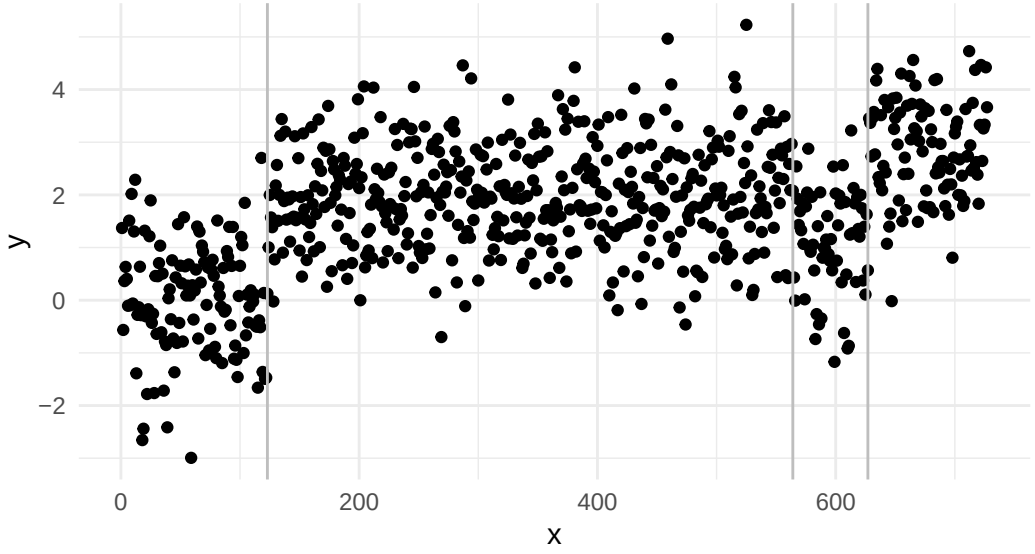
When dealing with real-world data and changepoint models, it's essential to evaluate how well the model fits the data. Apart from the techniques we saw last week, where we can use an elbow plot and visual inspection to assess a segmentation, this evaluation is often done by examining the residuals – the differences between the observed data points and the values predicted by the model. If the residuals exhibit certain patterns, it may indicate that the model is not capturing all the underlying structure of the data, or that assumptions about the error distribution are violated.

This section introduces three key diagnostic tools for assessing the residuals from a changepoint model: the **histogram of residuals**, the **normal Q-Q plot**, and the **residuals vs. fitted values plot**. These tools help assess whether the assumptions of the model (such as normality and homoscedasticity) hold.

As an example, we will run diagnostics on the following example:



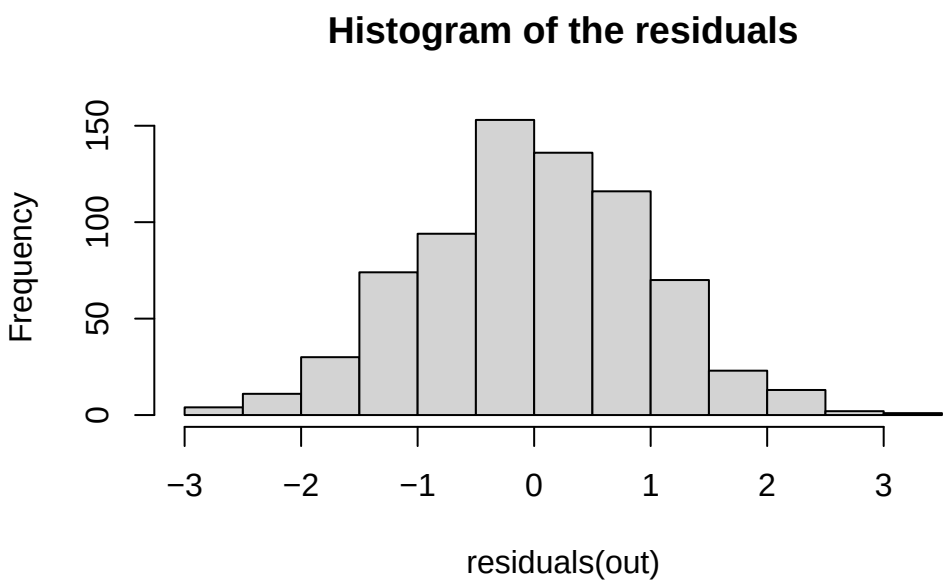
On which PELT returns the segmentation:



5.1.1.1 1. Histogram of the Residuals

The histogram of the residuals is a simple but effective tool for visualizing the distribution of residuals. The histogram checks whether the residuals are approximately normally distributed. You should see a bell-shaped histogram centered around zero, indicating that the residuals are symmetrically distributed around the mean with no significant skewness or heavy tails.

```r
hist(residuals(out), main = "Histogram of the residuals")
```

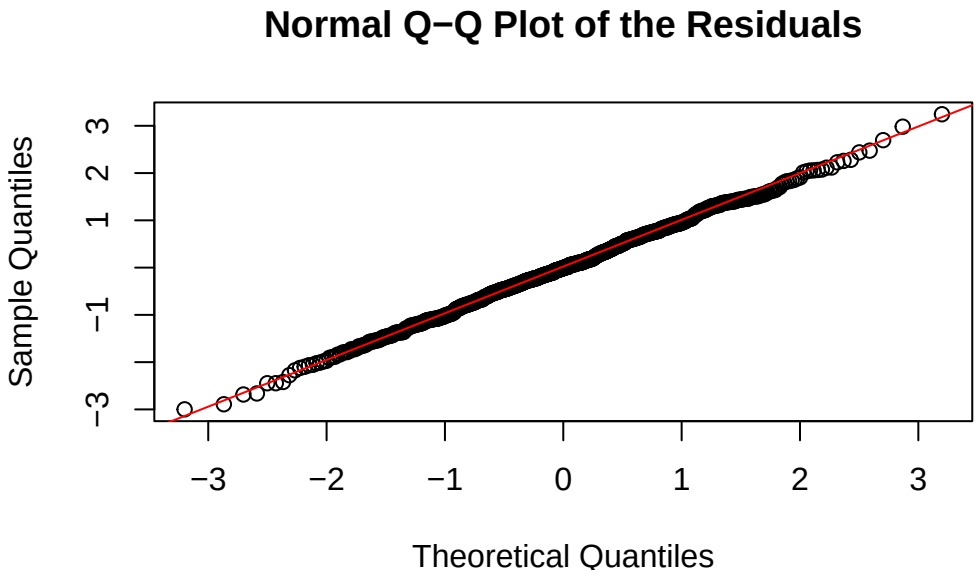


- **What to Watch for:**
  - Skewness: If the histogram is not symmetric, it could indicate skewness in the residuals, suggesting that the model may not fully capture the data's structure.
  - Outliers: Large bars far from zero indicate extreme residuals, which could be outliers.
  - Heavy or light tails: If the histogram shows long tails, it may suggest that the data contains more extreme values than expected under the assumption of normally distributed errors.

5.1.1.2 2. Normal Q-Q Plot

The normal quantile-quantile (Q-Q) plot compares the distribution of the residuals to a theoretical normal distribution. The idea is to see if the residuals deviate significantly from the straight line that would indicate normality. Points should fall along a straight diagonal line, indicating that the residuals closely follow a normal distribution.

```r
qqnorm(residuals(out), main = "Normal Q-Q Plot of the Residuals")
qqline(residuals(out), col = "red")
```



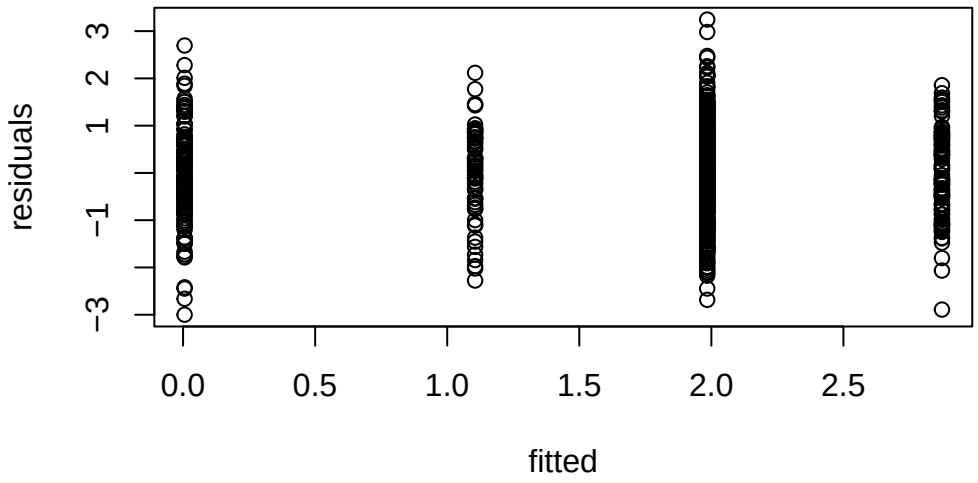
- **What to Watch for:**
  - **Deviations from the line:** Systematic deviations suggest non-normality. For instance, if points deviate upwards or downwards at the tails of the plot, it might indicate heavy tails (more extreme values than a normal distribution would predict) or light tails (fewer extreme values).
  - **Outliers:** Points far away from the line, especially at the ends, suggest outliers or non-normality in the tails.

The Q-Q plot is a more precise method of checking normality than the histogram since it directly compares the residuals to a normal distribution's quantiles.

5.1.1.3 3. Residuals vs. Fitted Values Plot

Finally, this plot shows the residuals on the y-axis against the fitted values from the model on the x-axis. It is particularly useful for checking if there are patterns in the residuals that suggest issues with the model fit. We would like to see a random scatter of points around zero, with no discernible pattern, and roughly equal spread across all fitted values. This suggests the model has adequately captured the relationship between the variables.

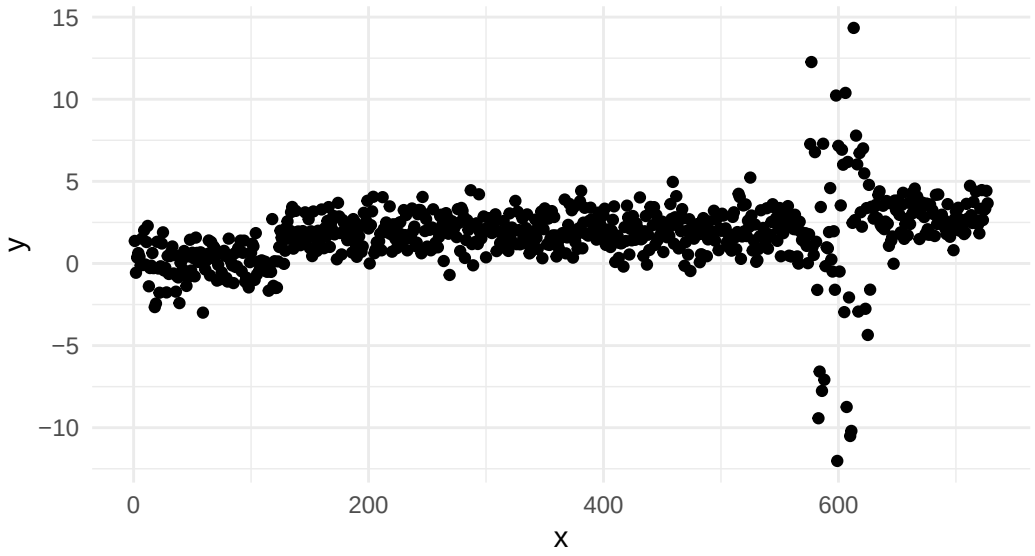
```
plot(fitted(out), residuals(out), xlab = "fitted", ylab="residuals")
```



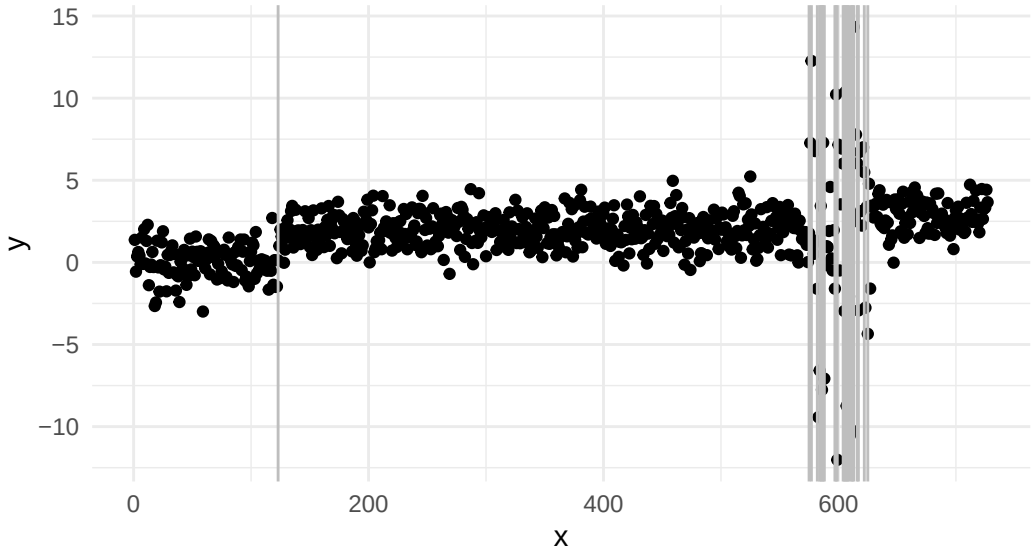
- **What to Watch for:**
  - To observe cluster of points in the data is normal, as the fitted values are the estimate of our signal. Maybe counter-intuitively, we need to look out for single observations alone! These could be segments which only have one or few observations in it, which could be a sign of overfitting.
  - Heteroscedasticity (Non-constant variance): If the residuals' spread increases or decreases as the fitted values increase, it may indicate heteroscedasticity, which violates one of the key assumptions of many models (constant variance of residuals). If you observe heteroschedasticity only in one of the clusters, it might mean that we are underestimating the number of the changes!

5.1.2 Example: violating heteroschedasticity:

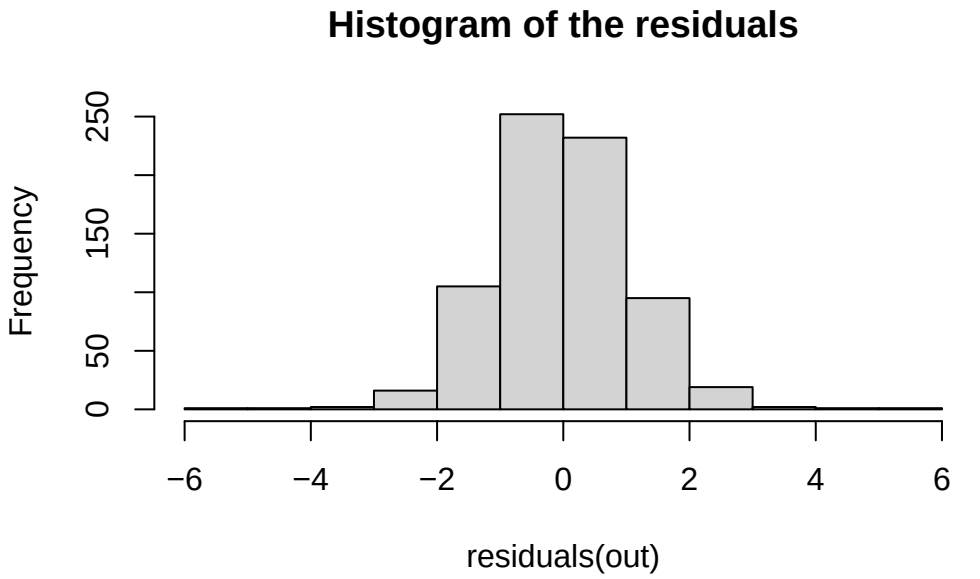
Let's take the data from before, and add increase the variance in one of the segments:

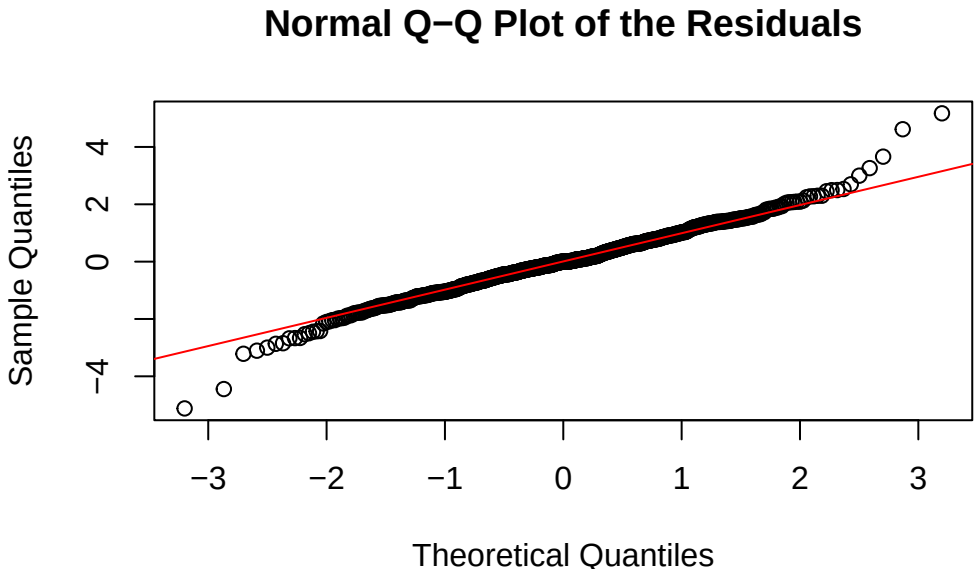


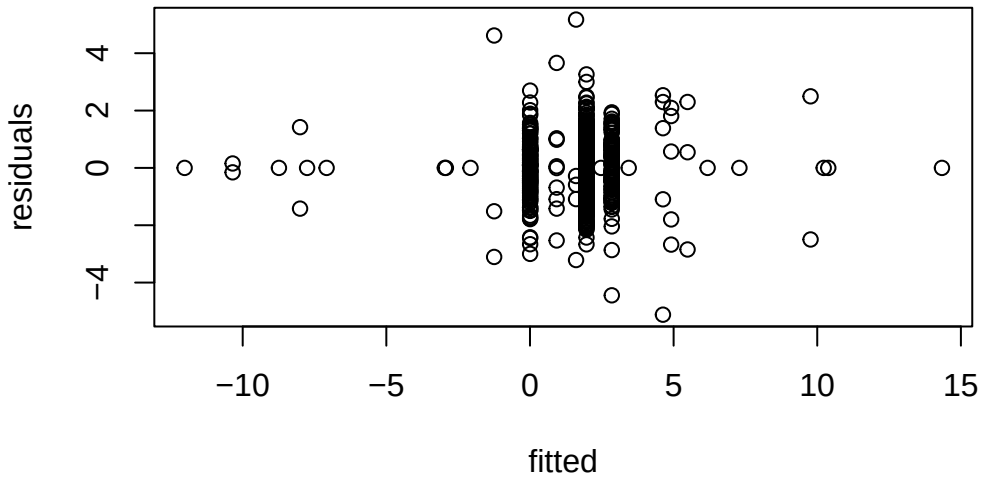
A simple PELT change-in-mean fit gives us the segmentation:



With the diagnostics:


Histogram of the residuals
Frequency
0
50
150
250
−6
−4
−2
0
2
4
6
residuals(out)


Normal Q−Q Plot of the Residuals
Sample Quantiles
−4
0
2
4
−3
−2
−1
0
1
2
3
Theoretical Quantiles



We can clearly see from the vertical lines that we have an oversegmentation in the third segment. The histogram and Q-Q plot both show some signs of deviations from a strict normal distribution, especially in the tails, which might be due to the presence of heteroscedasticity in the data. However, the residuals vs. fitted values plot provides the strongest evidence of overfitting due to heteroscedasticity, showing that the variance of the residuals is not constant and increases at the extremes.

5.2 Estimating Other Known Parameters

Let's revisit the classic problem of detecting a change in mean. One of the key assumptions we've relied on so far is that the variance, σ^2, is fixed, and known. Specifically, we used the following cost function in our models:

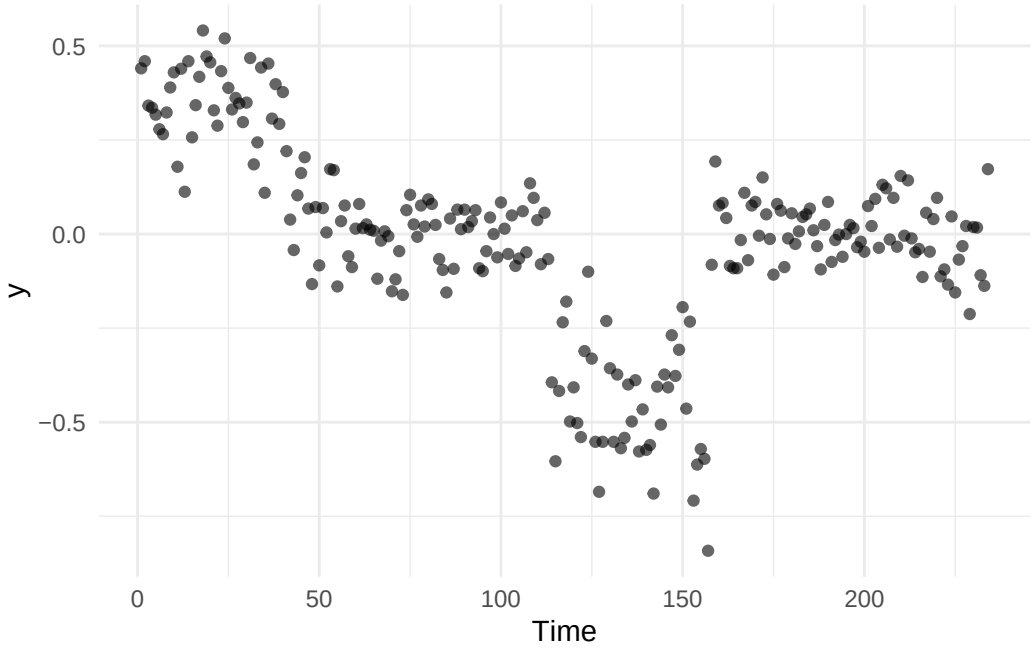
$$\mathcal{L}(y_{s:t}) = \frac{1}{2\sigma^2} \sum_{i=s}^{t} (y_i - \bar{y}_{s:t})^2$$

In our examples, we've typically set $\sigma^2 = 1$. However, this assumption is often unrealistic when working with real data. When the true value of σ^2 is unknown or incorrectly specified, the results of changepoint detection can be significantly affected.

- If we underestimate the variance by choosing a value for σ^2 that is too small, the changepoint detection algorithm may overlook real changes in the data, resulting in fewer detected changepoints.
- Conversely, if we overestimate the variance with a value that is too high, the algorithm may detect too many changes, identifying noise as changepoints.

5.2.1 Neuroblastoma Example: The Impact of Mis-specified Variance

Consider the neuroblastoma dataset as an example. If we run a changepoint detection method like PELT or BS on this data without any pre-processing, we might observe that the algorithm does not detect any changes at all:




```
summary(out_op)
```

```
Created Using changepoint version 2.3
Changepoint type      : Change in mean
Method of analysis    : PELT
Test Statistic  : Normal
Type of penalty       : MBIC with value, 16.36596
Minimum Segment Length : 1
Maximum no. of cpts   : Inf
Changepoint Locations :
```

In this example, PELT fails to detect any changes because the scale of the data suggests a lower variance than expected, affecting the algorithm's sensitivity to changes.

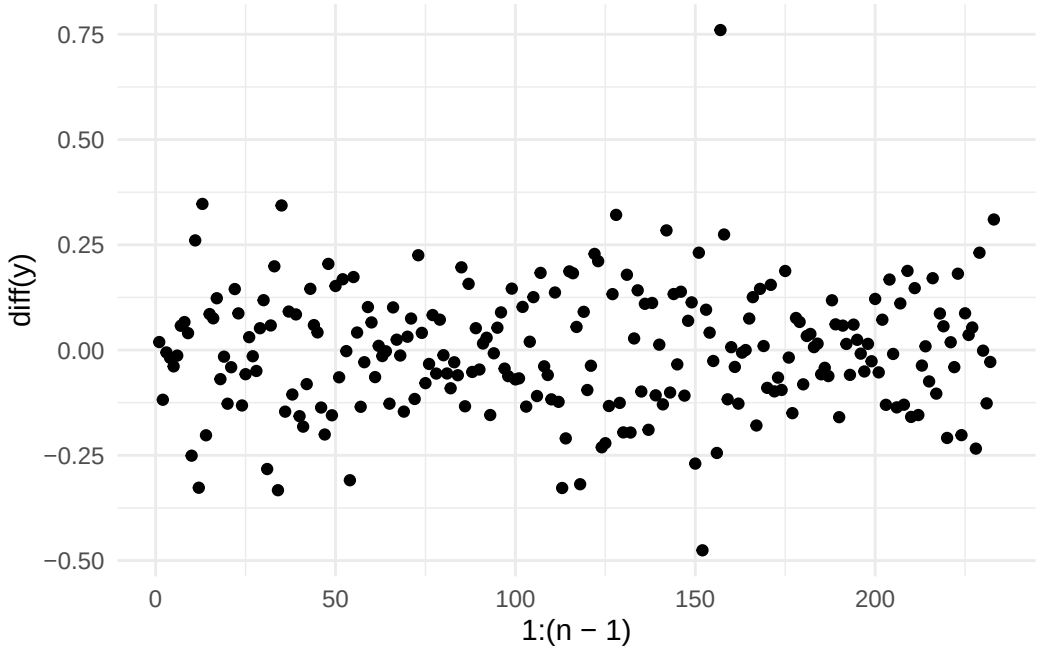
5.2.2 Addressing Mis-specified Variance with Robust Estimators

One problem with estimating the variance in the change-in-mean scenario, is that depending on the size of the changes, these can skew your estimate…

One way to solve the issue of this, is that, on the assumption that the data is i.i.d. Gaussian, looking at the lag-1 differences $z_t = y_t - y_{t-1} \; \forall \quad t = 2, \dots, n$:

```
qplot(x = 1:(n-1), y = diff(y)) + theme_minimal()
```

```
Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```



And compute the sample variance across all these differences as an estimator for our sigma square: $\hat{\sigma}^2 = \bar{S}(z_{1:n})$. However, we have not fixed our problem… yet!

What happens exactly at $t = \tau + 1$? Well, across these observations, our $z_{\tau+1}$ appears as an outlier (why?). This can still skew our estimate of the variance.

A solution, is to use robust estimators of the variance. A common choice is the Median Absolute Deviation (MAD), which is less sensitive to outliers and can provide a more reliable estimate of $\bar{S}$ in our case.

The formula for MAD is given by:

$$\text{MAD} = \text{median}(|z_i - \text{median}(z_{1:n})|)$$

This estimator computes the median of the absolute deviations from the median of the data.

However, for asymptotical consistency, to fully convert MAD into a robust variance estimate, we can use:

$$\hat{\sigma}_{\text{MAD}} = 1.4826 \times \text{MAD}$$

This scaling factor ensures that σ_{MAD} provides an approximately unbiased estimate of the standard deviation under the assumption of normally distributed data.

We then can divide our observations by this value to obtain ready-to-analyse observations. Go back and check the scale of the data in the segmentations in week 3!

While this trick provides a solution for handling variance estimation in the change-in-mean problem, more sophisticated models may require the estimation of additional parameters. And more advanced techniques are needed to ensure that all relevant parameters are accurately estimated (this is very much an open are of research)!

5.3 Non-Parametric Models

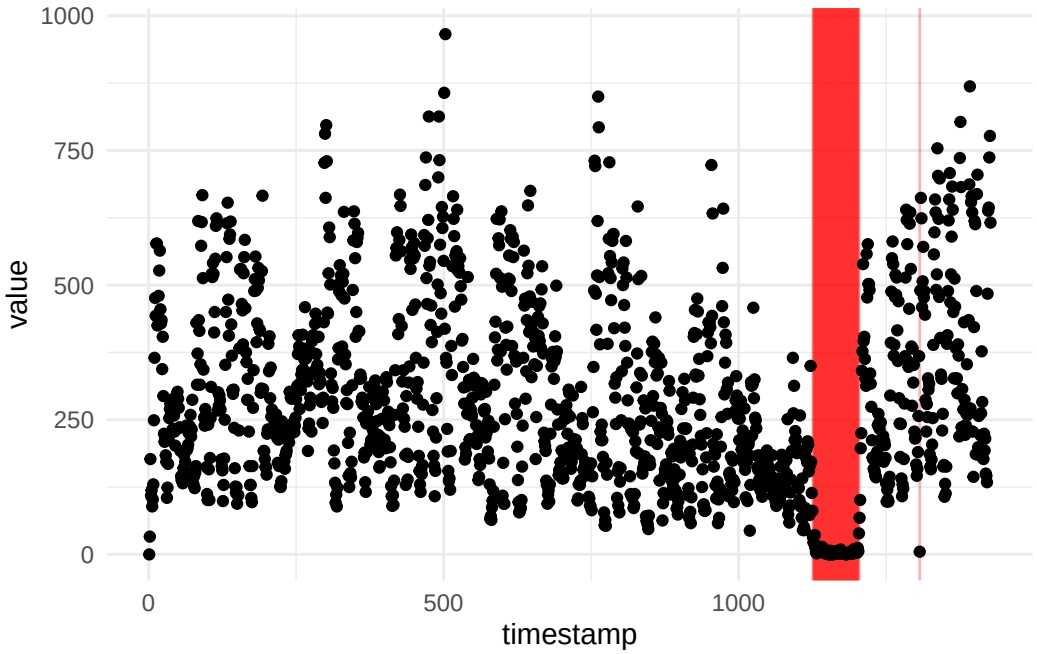
A alternative approach for detecting changes in real data, especially when we don't want to make specific parametric assumptions, is to use a non-parametric cost function. This method allows us to detect general changes in the distribution of the data, not just changes in the mean or variance. One such approach is the Non-Parametric PELT (NP-PELT) method, which focuses on detecting any changes in the underlying distribution of the data.

For example, let us have a look at one of the sequences from the Yahoo! Webscope dataset ydata-labeled-time-series-anomalies-v1_0 [http://labs.yahoo.com/Academic_Relations]:

```
A1 <- read_csv("extra/A1_yahoo_bench.csv")
```

```
Rows: 1427 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): timestamp, value, is_anomaly

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
ggplot(A1, aes(x = timestamp, y = value)) +
  geom_vline(xintercept = which(A1$is_anomaly == 1), alpha = .3, col = "red") +
  geom_point() +
  theme_minimal()
```



Following Haynes, Fearnhead, and Eckley (2017), we introduce the NP-PELT approach. Let $F_{i:n}(q)$ denote the unknown cumulative distribution function (CDF) for the segment $y_{1:n}$, where n indexes the data points. Similarly, let $\hat{F}_{1:n}(q)$ be the empirical CDF, which provides an estimate of the true distribution over the segment. The empirical CDF is given by:

$$\hat{F}_{1:n}(q) = \frac{1}{n}\left\{\sum_{j=1}^{n} \mathbb{I}(y_j < q) + 0.5 \times \mathbb{I}(y_j = q)\right\}.$$

Here, $\mathbb{I}(y_j < q)$ is an indicator function that equals 1 if $y_j < q$ and 0 otherwise, and the term $0.5 \times \mathbb{I}(y_j = q)$ handles cases where y_j equals q.

Under the assumption that the data are independent, the empirical CDF $\hat{F}_{1:n}(q)$ follows a Binomial distribution. Specifically, for any quantile q, we can write:

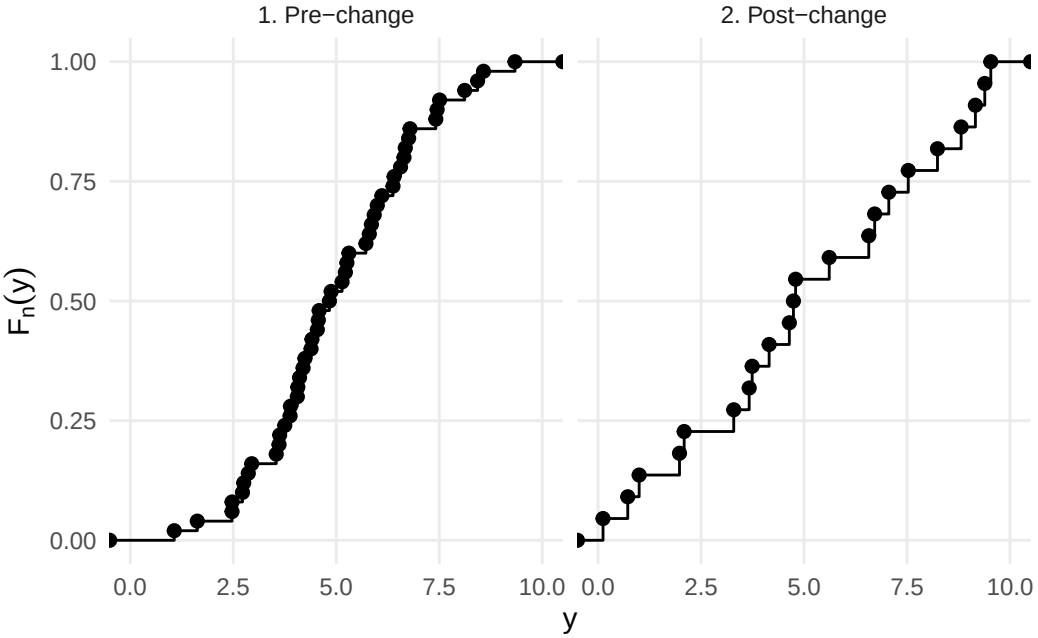
$$n\hat{F}_{1:n}(q) \sim \text{Binom}(n, F_{1:n}(q)).$$

This means that the number of observations y_j less than or equal to q follows a Binomial distribution, with n trials and success probability equal to the true CDF value $F_{1:n}(q)$ at q.

Using this Binomial approximation, we can derive the log-likelihood of a segment of data $y_{\tau_1+1:\tau_2}$, where τ_1 and τ_2 are the changepoints marking the beginning and end of the segment, respectively. The log-likelihood is expressed as:

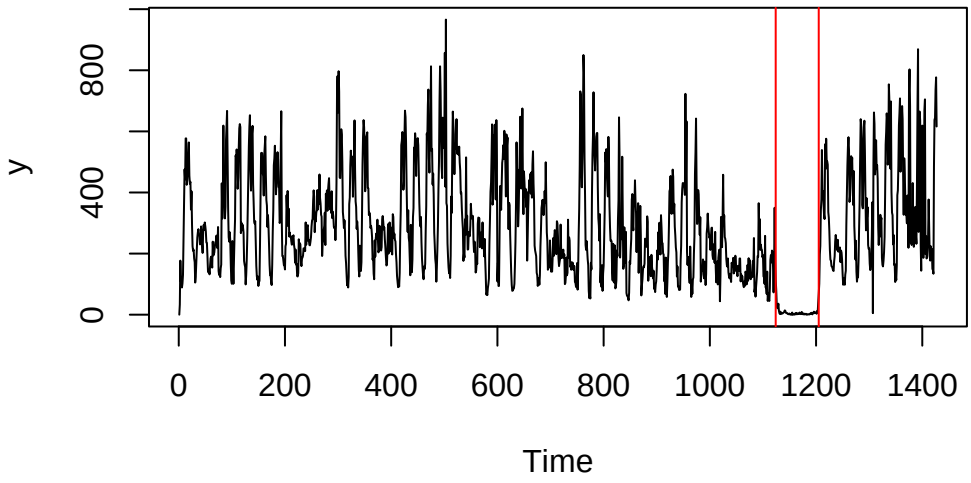
$$\mathcal{L}(y_{\tau_1+1:\tau_2}; q) = (\tau_2 - \tau_1)\left[\hat{F}_{\tau_1+1:\tau_2}(q)\log(\hat{F}_{\tau_1+1:\tau_2}(q)) - (1-\hat{F}_{\tau_1+1:\tau_2}(q))\log(1-\hat{F}_{\tau_1+1:\tau_2}(q))\right].$$

This cost function compares the empirical CDF of at the right and at the left of this data points, for all the points:



In practice, NP-PELT on the previous sequence gives the following:

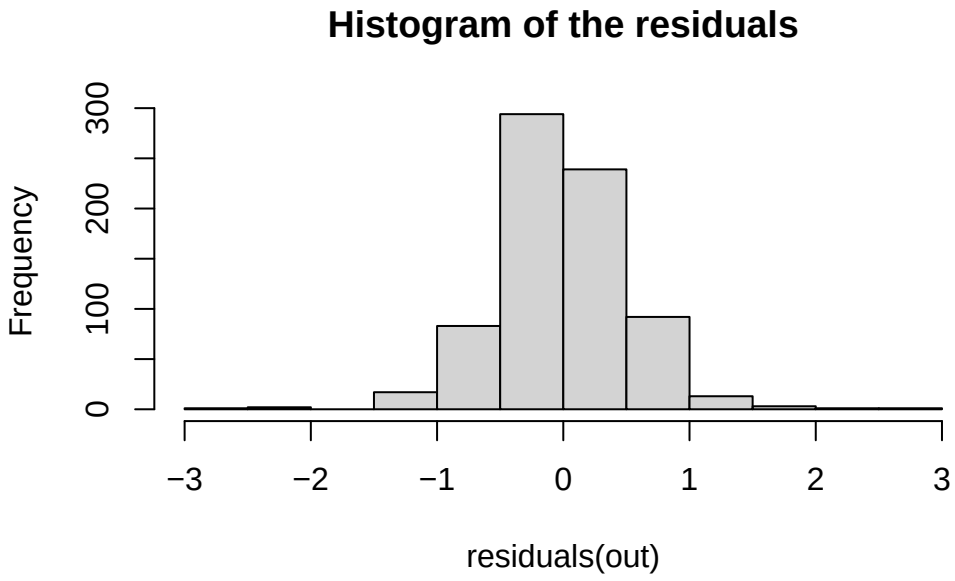
```
library(changepoint.np)

y <- A1$value

cpt.np(y, penalty = "Manual", pen.value = 25 * log(length(y))) |> plot(ylab = "y")
```

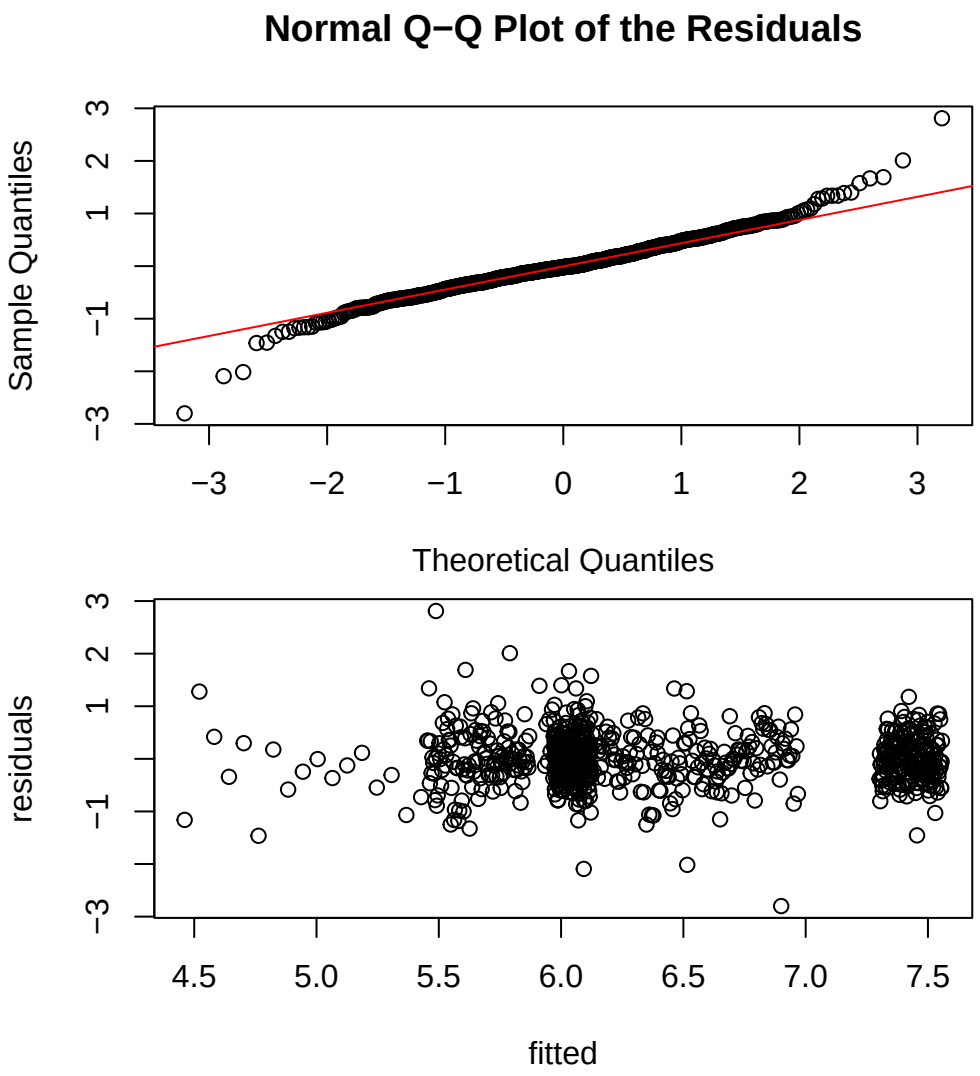


5.4 Exercises

5.4.1 Workshop exercises

Provide an interpretation of the residuals diagnostics from the Simpsons dataset:




Normal Q−Q Plot of the Residuals
Sample Quantiles
−3
−1
1
2
3
−3
−2
−1
0
1
2
3
Theoretical Quantiles
residuals
−3
−1
1
2
3
4.5
5.0
5.5
6.0
6.5
7.0
7.5
fitted

References

Baranowski, Rafal, Yining Chen, and Piotr Fryzlewicz. 2019. "Narrowest-over-Threshold Detection of Multiple Change Points and Change-Point-Like Features." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 81 (3): 649–72.

Bown, Jonathan. 2023. "Simpsons Episodes & Ratings (1989-Present)." https://www.kaggle.com/datasets/jonbown/simpsons-episodes-2016?resource=download.

Fryzlewicz, Piotr. 2014. "Wild binary segmentation for multiple change-point detection." *Annals of Statistics* 42: 2243–81.

Haynes, Kaylea, Idris A Eckley, and Paul Fearnhead. 2017. "Computationally Efficient Changepoint Detection for a Range of Penalties." *Journal of Computational and Graphical Statistics* 26 (1): 134–43.

Haynes, Kaylea, Paul Fearnhead, and Idris A Eckley. 2017. "A Computationally Efficient Nonparametric Approach for Changepoint Detection." *Statistics and Computing* 27: 1293–1305.

Jackson, Brad, Jeffrey Scargle, D. Barnes, S. Arabhi, A. Alt, P. Gioumousis, E. Gwin, P. Sangtrakulcharoen, L. Tan, and Tun Tsai. 2005. "An Algorithm for Optimal Partitioning of Data on an Interval." *Signal Processing Letters, IEEE* 12 (March): 105–8. https://doi.org/10.1109/LSP.2001.838216.

Killick, R., P. Fearnhead, and I. A. Eckley. 2012. "Optimal Detection of Changepoints with a Linear Computational Cost." *Journal of the American Statistical Association* 107 (500): 1590–98.

Scott, Andrew Jhon, and Martin Knott. 1974. "A Cluster Analysis Method for Grouping Means in the Analysis of Variance." *Biometrics*, 507–12.

Sen, Ashish, and Muni S Srivastava. 1975. "On Tests for Detecting Change in Mean." *The Annals of Statistics*, 98–108.

Yao, Yi-Ching, and Richard A Davis. 1986. "The Asymptotic Behavior of the Likelihood Ratio Statistic for Testing a Shift in Mean in a Sequence of Independent Normal Variates." *Sankhyā: The Indian Journal of Statistics, Series A*, 339–53.

Zhang, Nancy R, and David O Siegmund. 2007. "A Modified Bayes Information Criterion with Applications to the Analysis of Comparative Genomic Hybridization Data." *Biometrics* 63 (1): 22–32.